Weber
Türschmann

BASIC 2

2. Auflage

UTB

Haupt

Uni-Taschenbücher 589

UTB

Eine Arbeitsgemeinschaft der Verlage

Birkhäuser Verlag Basel und Stuttgart
Wilhelm Fink Verlag München
Gustav Fischer Verlag Stuttgart
Francke Verlag München
Paul Haupt Verlag Bern und Stuttgart
Dr. Alfred Hüthig Verlag Heidelberg
Leske Verlag + Budrich GmbH Opladen
J. C. B. Mohr (Paul Siebeck) Tübingen
C. F. Müller Juristischer Verlag – R. v. Decker's Verlag Heidelberg
Quelle & Meyer Heidelberg
Ernst Reinhardt Verlag München und Basel
F. K. Schattauer Verlag Stuttgart-New York
Ferdinand Schöningh Verlag Paderborn
Dr. Dietrich Steinkopff Verlag Darmstadt
Eugen Ulmer Verlag Stuttgart
Vandenhoeck & Ruprecht in Göttingen und Zürich
Verlag Dokumentation München

Karl Weber
Carl Wolfram Türschmann

BASIC 2

Lehr- und Handbuch der Programmiersprache BASIC
mit wirtschaftswissenschaftlichen
Anwendungsbeispielen

2., überarbeitete und erweiterte Auflage

Verlag Paul Haupt Bern und Stuttgart

Die Autoren:
Karl Weber, Dr. oec. publ., M. S. in Accountancy.
1948–1954 Studium der Volks- und Betriebswirtschaftslehre an der Universität Zürich; 1955 Promotion; 1953–1956 Assistent am Handelswissenschaftlichen Seminar der Universität Zürich; 1956–1963 Wissenschaftlicher Mitarbeiter am Betriebswissenschaftlichen Institut der ETH Zürich; 1959 Studium an der London School of Economics and Political Science; 1963–1964 Studium und Tätigkeit als Instructor of Accountancy an der University of Illinois in Urbana; 1964–1967 Assistant Professor of Accountancy an der University of Illinois; 1966 Habilitation an der Universität Zürich; 1966 Fellow in Accountancy an der University of Illinois; 1967–1969 Associate Professor of Accountancy an der University of Illinois; seit 1969 Professor für Betriebswirtschaftslehre an der Justus-Liebig-Universität Giessen.

Carl Wolfram Türschmann, Dipl.-Ing., Dipl.-Oec., Dr. rer. soc.
1964–1970 Studium des Maschinenbaus an der Technischen Hochschule Darmstadt; 1970–1973 Studium der Wirtschaftswissenschaften an der Justus-Liebig-Universität Giessen; 1970–1972 akademischer Tutor im Fachbereich Wirtschaftswissenschaft der Justus-Liebig-Universität; 1972–1977 wissenschaftlicher Mitarbeiter an der Professur für Betriebswirtschaftslehre V der Justus-Liebig-Universität Giessen; 1973–1976 freier Mitarbeiter an der Abteilung für experimentelle Kardiologie am Max-Planck-Institut für physiologische und klinische Forschung (W. G. Kerckhoff-Institut), Bad Nauheim; 1976–1980 Lehrbeauftragter für computergestützte Methoden der empirischen Sozialforschung an der Ruhr-Universität Bochum (Institut für Sportwissenschaft); 1978–1980 Forschungsbeauftragter und Projektleiter des Bundesministeriums für innerdeutsche Beziehungen an der Abteilung für Soziologie der Ruhr-Universität Bochum; 1979 Promotion; seit 1980 Hochschulassistent (Assistenzprofessor) für Betriebswirtschaftslehre an der Justus-Liebig-Universität Giessen.

CIP–Kurztitelaufnahme der Deutschen Bibliothek

Weber, Karl:
BASIC: Lehr- u. Handbuch d. Programmiersprache
BASIC mit wirtschaftswiss. Anwendungsbeispielen / Karl Weber; Carl Wolfram Türschmann. – Bern; Stuttgart: Haupt
NE: Türschmann, Carl Wolfram:
2. – 2., überarb. u. erw. Aufl. – 1982. (Uni-Taschenbücher; 589)
ISBN 3–258–03175–4
NE: GT

Copyright © 1982 by Paul Haupt Berne
Alle Rechte vorbehalten. Jede Art der Vervielfältigung ohne Genehmigung des Verlages ist unzulässig.
Printed in Germany

Vorwort zur zweiten Auflage

Das vorliegende Lehr- und Handbuch soll als BASIC-Einführung und zur BASIC-Vertiefung dienen. Während in Band 1 die BASIC-Elementaranweisungen behandelt werden, befasst sich Band 2 mit den BASIC-Sonderanweisungen. Matrizen-, File- und erweiterte Programmsteuerungsanweisungen werden unter Bezugnahme auf ANSI BASIC 79 und unter Mitberücksichtigung von Neuregelungen gemäss ANSI BASIC 82 dargestellt. Zur Illustration der theoretischen Ausführungen dienen zahlreiche Beispiele. Das Buch vermittelt weiterhin einen Überblick über Minicomputersysteme mit BASIC-Prozessoren.
Die behandelten BASIC-Sonderanweisungen können mit dem an der Professur für Betriebswirtschaftslehre V seit 1977 weiterentwickelten FOSBIC-Compilersystem in weitestgehendem Masse ANSI BASIC entsprechend verarbeitet werden. Die FOSBIC Version 2 ist an der Justus-Liebig-Universität Giessen seit 1980 im praktischen Einsatz und wird in absehbarer Zeit zur allgemeinen Benutzung freigegeben.

Kontaktadresse:
Professur für Betriebswirtschaftslehre V
Justus-Liebig-Universität Giessen
Licher Strasse 74
D-6300 Giessen

Karl Weber
Carl Wolfram Türschmann

Giessen 1982

Vorwort zur ersten Auflage

Das vorliegende Lehr- und Handbuch zur Programmiersprache BASIC stellt das Ergebnis von Lehr- und Forschungstätigkeiten der Professur für Betriebswirtschaftslehre V der Justus-Liebig-Universität Giessen dar.
Die Arbeit wurde durch enge Kontakte mit amerikanischen Hochschulen wesentlich gefördert. Als wegweisend erwiesen sich die am Dartmouth College und an der University of Washington durchgeführten BASIC-Entwicklungen und die darüber orientierenden Arbeitspapiere, die den Verfassern zugänglich waren.
In verdankenswerter Weise stellten die Computerherstellfirmen ihre BASIC-Manuale zur Auswertung zur Verfügung und beteiligten sich auch an einer BASIC-Enquete.
Die Entwicklung des Lehrbuchs wurde weiterhin durch rege Teilnahme der Studenten an BASIC-Kursen und insbesondere auch durch Verwendung der Programmiersprache BASIC in Diplomarbeiten aus dem Gebiet des Operations Research gefördert.
Besonderer Dank gehört den Mitarbeitern der Professur für Betriebswirtschaftslehre V, insbesondere den Herren Dipl.-Math. Richard Trzebiner, cand. rer. pol. Bernd Stiehl und Dipl.-Oec. Horst Tempelmeier sowie Frau Sieglinde Stitz, für tatkräftige Mithilfe bei der Überarbeitung und Fertigstellung des Manuskripts.

Karl Weber
Carl Wolfram Türschmann — Giessen 1977

Inhaltsverzeichnis

Einleitung 11
1 BASIC-Charakteristik 12
11 BASIC-Sprachsystem 12
12 BASIC-Minicomputersysteme 13
121 Allgemeines 13
122 Repräsentativsysteme 14
122.1 Wang Laboratories, Inc. 14
122.2 Data General Corporation 15
122.3 Digital Equipment Corporation 16
122.4 Hewlett-Packard Company 18
122.5 IBM Corporation 19
122.6 Prime Computer, Inc. 20
122.7 Heinrich Dietz 20
122.8 Nixdorf Computer AG 20
122.9 Ing. C. Olivetti & C.S.p.A. 21
123 Realisierter BASIC-Sprachumfang 25
13 BASIC-Hochschuleinsatz 25
131 Allgemeines 25
132 Benutzerprogramme 28
133 FOSBIC-Compilersystem 29
14 BASIC-Standardisierung 30
2 BASIC-Problembearbeitung 34
21 Problembeschreibung und Problemanalyse 34
22 Programmplanung 35
23 Programmierung 39
24 Programmalternativen 43
3 BASIC-Struktur 45
31 Allgemeines 45
32 Elementaranweisungen 47
33 Sonderanweisungen 48
331 Matrizenanweisungen 49
331.1 MAT-Eingabeanweisungen 51
331.11 MAT READ-Statements 51
331.111 DIM-gebundenes MAT READ-Statement . 53

331.112 DIM-begrenztes MAT READ-Statement . . 58
331.113 DIM-umfassendes MAT READ-Statement . 66
331.12 MAT INPUT-Statements 68
331.2 MAT-Wertzuordnungsanweisungen 73
331.21 MAT LET-Statements 74
331.211 Numerisches MAT LET-Statement 74
331.212 Alphanumerisches MAT LET-Statement . . 90
331.22 MAT Let-Statements 92
331.221 Numerisches MAT let-Statement 93
331.222 Alphanumerisches MAT let-Statement . . . 103
331.223 Graphisches MAT let-Statement 105
331.3 MAT-Ausgabeanweisungen 106
331.31 MAT PRINT-Statements 106
331.311 DIM-gebundenes MAT PRINT-Statement . 110
331.312 DIM-begrenztes MAT PRINT-Statement . . 112
331.32 MAT PRINT USING-Statements 114
332 Fileanweisungen 116
332.1 Organisation und Struktur von Files 119
332.11 Filestruktur 119
332.111 Zeichenstrukturierte Files 119
332.112 Inhaltstrukturierte Files 120
332.12 Fileorganisation 121
332.121 Sequentielle Fileorganisation 122
332.122 Relative Fileorganisation 123
332.2 Filevereinbarungen 123
332.21 OPEN FL-Statement 124
332.22 CLOSE FL-Statement 132
332.23 SCRATCH FL-Statement 133
332.3 Fileausgabeanweisungen 135
332.31 OUTPUT FL-Statement 135
332.32 OUTPUT USING FL-Statement 143
332.33 MAT OUTPUT FL-Statement 145
332.34 MAT OUTPUT USING FL-Statement . . . 147
332.4 Fileeingabeanweisungen 147
332.41 INPUT FL-Statement 148
332.42 MAT INPUT FL-Statement 152
332.5 Filekontrollanweisungen 156
332.51 RESTORE FL-Statement 157
332.52 INQUIRE FL-Statement 161
332.53 IF EOF-Statement 164
332.6 Fileanweisungen verschiedener BASIC-Programmiersysteme 165
333 Erweiterte Programmsteuerungsanweisungen 168

333.1 Subroutineanweisungen 168
333.11 CALL-Statement 169
333.12 SUB-Block 173
333.2 Programmsegmentierungsanweisungen 181
333.21 CHAIN-Statement 182
333.22 LINK-Statement 182

Einleitung

Die drei BASIC-Bände sind als geschlossenes Ganzes zu sehen. Als Schnittstelle zu Band 1 dient der Übergang von den BASIC-Elementaranweisungen zu den BASIC-Sonderanweisungen.

Die in Band 1 behandelten BASIC-Elementaranweisungen beinhalten im wesentlichen den vom American National Standards Institute (ANSI) und von der European Computer Manufacturers Association (ECMA) für Minimal BASIC vorgegebenen Sprachumfang.

In Band 2 werden die Matrizen-, File- und erweiterten Programmsteuerungsanweisungen behandelt. Sie gehören – zusammen mit den in Band 3 dargestellten – Graphik- und Diagnoseanweisungen zu den BASIC-Sonderanweisungen.

Zur Beschreibung der im Text häufig erwähnten FOSBIC Version 1 vergleiche man:

> Karl Weber und Carl Wolfram Türschmann. FOSBIC-Compiler. Ein BASIC-Compilersystem auf FORTRAN-Basis. Bern 1977.

Die BASIC-Standardisierungsvorschläge des American National Standards Institute sind enthalten in:
ANSI. American National Standard for BASIC. Preliminary Draft. September 1979. Hanover 1979 und ANSI. American National Standard for BASIC. Draft Prosposed. February 15, 1982. Hanover 1982.

1 BASIC-Charakteristik

11 BASIC-Sprachsystem

Die zum Aufbau von BASIC-Programmen dienenden Statements werden als BASIC-Sprachanweisungen bezeichnet. Sie lassen sich in Elementar- und Sonderanweisungen einteilen.

$$\left\langle \begin{array}{l}\text{BASIC-Sprach-}\\ \text{anweisung}\end{array} \right\rangle_{204} := \left\langle \begin{array}{l}\text{Elementar-}\\ \text{anweisung}\end{array} \right\rangle_{206} \mid \left\langle \begin{array}{l}\text{Sonder-}\\ \text{anweisung}\end{array} \right\rangle_{207 \mid 384}$$

Im Rahmen dieser Arbeit werden die BASIC-Elementaranweisungen in Band 1 behandelt; BASIC-Sonderanweisungen gelangen in Band 2 und Band 3 zur Darstellung. Vgl. Abbildung 1.

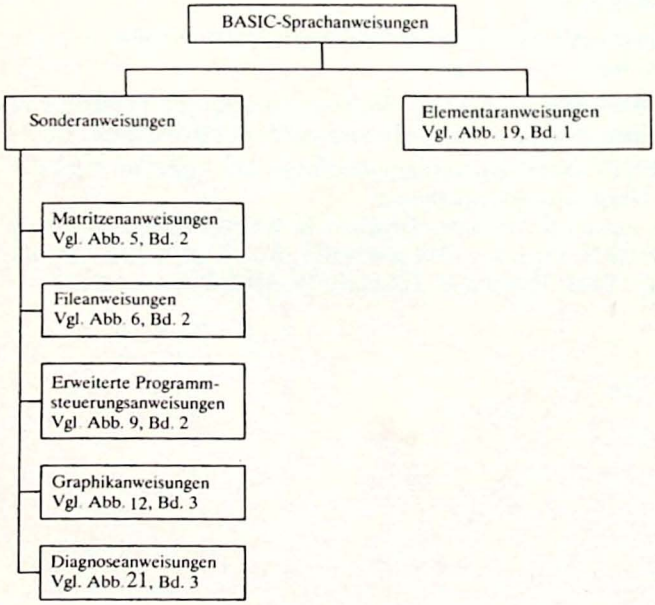


Abbildung 1.

12 BASIC-Minicomputersysteme

121 Allgemeines

Als wesentliches Merkmal von Kleincomputern wurde in den 60er Jahren vielfach das Fehlen von Compilern für höhere Programmiersprachen hervorgehoben.[1] Seitdem aber auch für Minicomputer in zunehmendem Masse leistungsfähige ALGOL-, BASIC-, FORTRAN- und/oder RPG-Compiler verfügbar sind, werden vor allem quantitative Merkmale zur Differenzierung zwischen Gross- und Kleincomputern verwendet, wobei insbesondere Wortlänge, Grösse des Kernspeichers, verfügbare Hardware und Anlagenpreis als Abgrenzungskriterien benutzt werden.[2]
In diesem Sinne werden – in einem Überblicksartikel der Zeitschrift Datamation aus dem Jahre 1969 – Kleincomputer generell als "small-word-length (i.e., 8-, 12-, 16 and 18-bit) numerical computers... priced in the order of $ 50 000 or less for minimum system and... offered with a normal complement of input/output equipment" umschrieben und die damals in den USA marktgängigen Anlagen weiterhin als "binary, fixedpoint, general-purpose machines with negative numbers represented in one's or two's complement notation" charakterisiert.[3]
Die seitherige Entwicklung auf dem Minicomputermarkt lässt erkennen, dass insbesondere der Preis kaum noch als Klassifikationsmerkmal für Anlagen der mittleren Datentechnik verwendbar ist.[4] Ähnliches gilt auch bezüglich der verfügbaren Hardware. So sind manche Maschinentypen hinsichtlich ihrer kleinsten Ausstattung sicherlich als Kleincomputer, bezüglich der – nach der Baukastentechnik erreichbaren – maximalen Ausstattung aber als Grosscomputer anzusprechen.
Unter Berücksichtigung der Entwicklungstendenzen in der mittleren Datentechnik scheint die folgende Definition vertretbar: Minicomputer sind vom Benutzer leicht bedienbare programmgesteuerte elektronische Rechner mit maximal 512 K Byte Kernspeicher bei maximaler Wortlänge von 36 Bits, deren Hardware unabhängig von externen Klimaanlagen installierbar ist.

1 Vgl. etwa J.-H. Bernhard. Klein-Computer, Würzburg 1972, S. 5.
2 Vgl. hierzu: Arno Schulz. Informatik für Anwender. Berlin 1973, S. 313; Carl Schneider. Kleincomputer oder Rechenzentrum. Wiesbaden 1972, S. 24.
3 D. J. Theis and L. C. Hobbs. Mini-Computers for Real-Time Applications. Datamation 15 (1969) 3, March, p. 39, 47.
4 Vgl. Paul B. Demitriades. Mini Update. Journal of Systems Management 25 (1974) 12, December, p. 31–32.

Beachtenswert ist, dass die marktgängigen Kleincomputer fast durchwegs mit BASIC-Compilern ausgerüstet sind.[5]

BASIC wird auch bei Minicomputern als höhere anwendungsorientierte Programmiersprache eingesetzt.

Dies trifft insbesondere für Minicomputersysteme zu, die im Timesharing-Betrieb verwendet werden.[6]

122 Repräsentativsysteme

122.1 Wang Laboratories, Inc.

Das BASIC-System Wang 2200 ist mit einem festverdrahteten BASIC-Compiler ausgerüstet. Es besteht aus einem Rechner und mehreren Ein- und Ausgabegeräten. Zur manuellen Befehlseingabe dient eine BASIC-Tastatur, die 24 BASIC-Kennworte und 12 BASIC-Standardfunktionen umfasst.[7] Maschinentechnisch ist beachtenswert, dass die BASIC-Statements im allgemeinen nur eine einzige Speicherzelle belegen (Einschrittbefehle).
Zur Grundausstattung des Systems Wang 2200 gehört auch ein CRT-Sichtgerät mit einer Kapazität von 16 Zellen à 64 Zeichen. Auf dem CRT-Monitor erscheinen die BASIC-Statements, die über die BASIC-Tastatur eingegeben werden, eventuelle Fehleranzeigen und schliesslich

5 Vgl. L. C. Hobbs and Richard A. McLaughlin. Minicomputer Survey. Datamation 20 (1974) 7, July, p. 52; Steve A. Bobick, Edmund J. Armon and Arthur W. Yerkes. Survey of Small Business Computers. Datamation 22 (1976) 10, October, p. 91–103, 105. – Über Minicomputer im allgemeinen: Infotech Information Limited. Minicomputers: State of the Art Report. Maidenhead 1973; Datapro Research Corporation. All About Minicomputers. Delran 1976.

6 Stephen A. Kallis, Jr. Mini, midi, vici. Data Management 14 (1976) 2. February, p. 27: "Some minicomputer systems have timesharing configurations. In such arrangements, as many as 16 terminals may run 'simultancously' for users programming in the same language, usually BASIC."

7 BASIC-Kennworte: COM, DATA, DEFFN, DIM END, FOR, GOSUB, GOTO, IF, INPUT, NEXT, PRINT, PRINT USING, READ, REM, RESTORE, RETURN, SELECT, STEP, STOP, TAB, THEN, TO, TRACE. BASIC-Funktionen: SIN, COS, TAN, ATN, EXP, LOG, ABS, SORT, RND, INT, SGN, PI. Wang Laboratories. Wang 2200 Datenblatt. Frankfurt [1972].

die Ergebnisse der durchgeführten Berechnungen. Eine Magnetbandkassetteneinheit dient zur Archivierung von Daten und Programmen. Damit besteht beim System Wang 2200 auch die Möglichkeit, ein bestimmtes Programm oder einen bestimmten Programmblock später vom Band in den Speicher zu laden und Programme ohne manuellen Eingriff zu verketten. Als Ausgabegeräte lassen sich Schnelldrucker, Ausgabeschreibmaschine und Plotter an das System anschliessen.[8]
Zur Dateiverarbeitung stehen Magnetband- und Magnetplatteneinheiten mit den entsprechenden BASIC-Statements zur Verfügung.
Bemerkenswert erscheint auch der Hinweis, dass über spezielle – allerdingst höchst inkompatible – maschinenorientierte Befehle, die zusammen mit BASIC-Statements verwendet werden können, der Zugriff auf einzelne Bits und deren Verarbeitung zugelassen wird.

122.2 Data General Corporation

Im Gegensatz zum BASIC-System Wang 2200 verfügen die Anlagen der DATA General Corporation nicht über einen festverdrahteten, dafür aber über einen besonders leistungsfähigen BASIC-Compiler.[9]

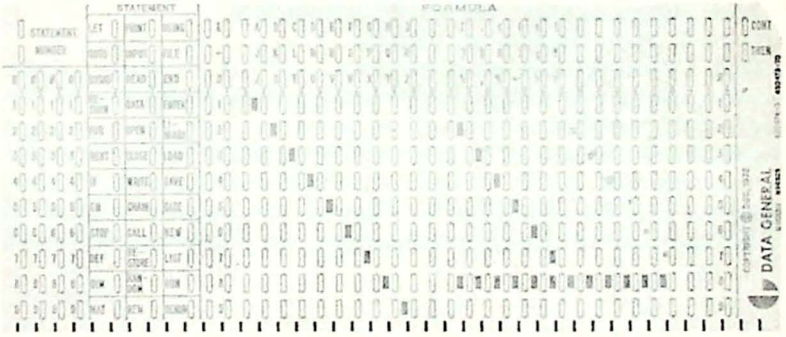


Abbildung 2. Markierungsbeleg für DG Extended BASIC

8 Wang Laboratories. 2200 BASIC-Language Specifications. Tewksbury 1973; Wang Laboratories. Wang BASIC Language Reference Manual. Lowell 1976.
9 Data General Corporation. An Introduction to BASIC. Southboro 1973; Data General Corporation. Extended BASIC User's Manual Southboro 1979.

DG Extended BASIC kann sowohl im Monologbetrieb (Stand-alone Operating System, SOS) als auch im Dialogbetrieb (Real Time Disk Operating System, RDOS) verwendet werden, wobei in beiden Fällen die Möglichkeit zum Teilnehmerbetrieb (Multi-user System) besteht.
DG Extended BASIC ist für den Monologbetrieb deshalb besonders geeignet, weil die Dateneingabe auch mit Hilfe von Markierungsbelegen (mark sense cards) erfolgen kann, über deren Aufbau Abbildung 2 orientiert. Im übrigen weist DG Extended BASIC folgende Besonderheiten auf:

- FORTRAN/ALGOL-Kompatibilität über Dateien,
- Benutzungsmöglichkeit von Assembler-Subroutinen, mit CALL,
- flexible Formatierung, mit COBOL-ähnlichen Format-Strings.

122.3 Digital Equipment Corporation

Von der Digital Equipment Corporation wurden verschiedene BASIC-Versionen entwickelt.
Für Anlagen vom Typ DEC PDP-8 steht BASIC-8 zum Teilnehmerbetrieb zur Verfügung. Diese Version ist mit Dartmouth BASIC-1 kompatibel, weist aber im Vergleich zu leistungsfähigeren BASIC-Varianten insbesondere folgende Restriktionen auf:[10]

- Keine Matrizenoperationen,
- keine alphanumerische Konstantenverarbeitung,
- keine implizite Felddimensionierung,
- keine mehrzeilige Benutzerfunktionen,
- maximale Programmlänge ca. 300 Statements, abhängig von der Programmkomplexität.

Ausser BASIC-8 kann auf Anlagen vom Typ DEC PDP-8 auch die leistungsfähigere IK BASIC-Version verwendet werden.[11] Ebenfalls verfügbar ist das DEC EduSystem 30. Dieses bietet insbesondere auch die Möglichkeit zur BASIC-Programmierung mit Markierungsbelegen.[12]

10 Digital Equipment Corporation. PDP-8 Family User's Guide. TSS/8 Time-Sharing System. Maynard 1970, p. 3.5–3.8; Digital Equipment Corporation. Introduction to Programming. PDP-8 Family Computers. 2nd ed., Maynard 1970, p. 10.23–10.31.

11 Vgl. Digital Equipment Corporation. Programming Languages. 2nd ed., Maynard 1972, p. 12.1–12.67. – Vgl. auch: Digital Equipment Corporation. Introduction to BASIC. Maynard 1978.

12 Digital Equipment Corporation. EduSystem Handbook. Maynard 1973, p. 7.1–7.52.

Abbildung 3 zeigt den Aufbau der bei diesem System zu verwendenden Lochkarten, deren Kennung (unter Schablonenverwendung) aufgrund der am oberen Kartenrand handschriftlich angegebenen BASIC-Statements (sehr zeitaufwendig) vorzunehmen ist.[13] Neben einer Reihe von Systembefehlen (NEW, RUN, OLD usw.) können folgende BASIC-Kennworte markiert werden: PRINT, REM, LET, IF, FOR, NEXT, READ, DATA, GOSUB, RETURN, GOTO, INPUT, DIM, WRITE, RESTORE, CHAIN, STOP, END, RANDOM.

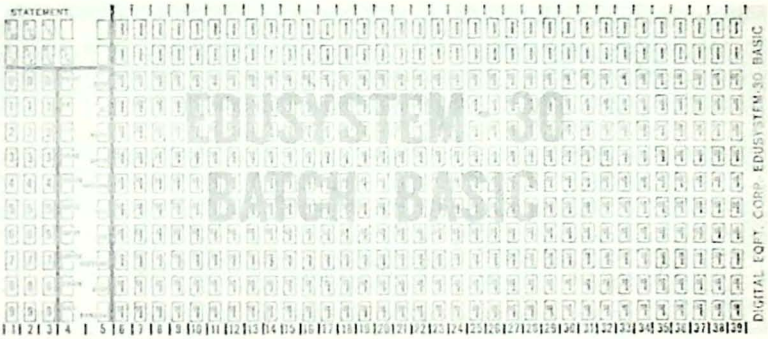


Abbildung 3. Markierungsbeleg für EduSystem30

Anlagen vom Typ DEC PDP-11 lassen die Verwendung von BASIC-Plus zu. Diese besonders leistungsfähige BASIC-Variante umfasst auch die folgenden Statements: IF/THEN/ELSE, FOR/UNLESS, FOR/WHILE, FOR/UNTIL, CHAIN, DEF multiple line, PRINT USING, ON ERROR GOTO, RESUME.[14]
Als weitere BASIC-Variante sei BASIC/RT 11 erwähnt. Sie bietet erfahrenen Programmierern die Möglichkeit zur Mitverwendung bereits

13 Digital Equipment Corporation. EduSystem 30 User's Guide. Mark-Sense Batch. Maynard 1971.

14 Digital Equipment Corporation. BASIC-Plus Language Manual for Use with RSTS-11 (PDP-11 Resource Time-Sharing-System). 3rd ed., Maynard 1973; Digital Equipment Corporation. PDP-11 Resource Time-Sharing System (RSTS-11) User's Guide. BASIC-Plus Programming Language. Maynard 1971. – Vgl. auch: Digital Equipment Corporation. PDP-11. BASIC-11. Language Reference Manual. Maynard 1976; Digital Equipment Corporation. PDP-11. Software Handbook. Maynard 1980.

vorliegender Assembler-Programme.[15] Weiterhin können – falls zur Datenerfassung und -Verarbeitung entsprechende Anlagen verfügbar sind – die Spezialanweisungen des Industrial BASIC benutzt werden.[16]

122.4 Hewlett-Packard Company

Die Hewlett-Packard Company übernahm 1968 Dartmouth BASIC-1 für Kleincomputer vom Typ H-P 2114, H-P 2115 und H-P 2116. Dabei wurde Dartmouth BASIC zunächst nur dahingehend modifiziert, dass (neben dem WAIT-Befehl zur Verzögerung des Programmablaufs) auch die Möglichkeit zum Aufruf von Assembler-Subroutinen (mit CALL) bestand.[17] Spätere Weiterentwicklungen dienten insbesondere zur komfortableren Verarbeitung von Zeichenketten (strings) und Dateien (files).[18]

Die modernen H-P Computer sind sowohl für Dialog- als auch für Monologbetrieb geeignet; mit der Anlage H-P 2100A (mit 16 bit Wortlänge und 8 K Kernspeicher) sind als Ein- und Ausgabegeräte verwendbar: Fernschreiber (10 Zeichen/s), Lochstreifenleser (400 Zeichen/s), Lochstreifenstanzer (75 Zeichen/s) und Mark/Sense-Kartenleser (200 Karten/min).

Der optische Mark/Sense-Kartenleser wird – auch nach deutschen Erfahrungen – mit Vorteil bei BASIC-Applikationen im Unterrichtsbereich verwendet. Begründung: „Der Student wird von mechanischen Stanzgeräten, die wegen der Kosten nie in der nötigen Anzahl vorhanden sind, unabhängig und kann sein Programm mit dem Stift zu Hause herstellen. Damit ist der Programmierunterricht auf breiter Front möglich."[19] Speziell zu erwähnen sind auch die (mit einem umfangreichen Graphiksystem ausgerüsteten) HP Systeme 35 und 45.[18a]

15 Digital Equipment Corporation. BASIC/RT11 Language Reference Manual. Maynard 1973.

16 Digital Equipment Corporation. Industrial BASIC – Solving the Industrial Control Problem. Maynard 1974. – Vgl. auch: J. R. Brandenberger. Formal Instruction in Lab Computing. Datamation 20 (1974) 6, June, p. 49–52.

17 Hewlett-Packard. BASIC Language Reference Manual for Hewlett-Packard 2114A/B, 2115A and 2116B Computers. Palo Alto 1968 [= Originalversion des Dartmouth BASIC-1 Handbuchs, mit HP-spezifischem Anhang].

18 Hewlett-Packard. 2000A: A Guide to Time Shared BASIC for Reference and Self-Instruction. Cupertino 1970; Hewlett-Packard. Time-Sharing BASIC. Nieder Eschbach 1970.

18a Hewlett-Packard. System 35 Tischcomputer. Bedienungs- und Programmier-Handbuch. Böblingen 1980; Hewlett-Packard. System 45. Program-

Die Hewlett-Packard Company bemühte sich in Amerika auch speziell um die Entwicklung computergestützter Unterrichtsmethoden. Verwiesen sei insbesondere auf die Beteiligung von HP am Projekt Solo (der Universität Pittsburgh), die Übernahme von Simulationsprogrammen (MARKET und POLICY für H-P 2000E/F)[20], die BASIC-programmierten Marketing-Fallstudien[21] sowie auf die umfangreichen Programmsammlungen aus dem Gebiete von Mathematik, Statistik, Operations Research und allgemeiner Betriebswirtschaftslehre.[22]

122.5 IBM Corporation

Der Minicomputer IBM 5100 weist in der Grundausstattung eine Kernspeichergrösse von 16 K Zeichen (characters), einen Kleinbildschirm (16 Zeilen à 64 Zeichen) und eine Magnetbandeinheit (206 K) auf. Die Kernspeichergrösse kann – in Stufen von 16 K – bis maximal 64 K ausgeweitet werden; anschliessbar sind ein Drucker (132 Druckstellen, 80 Zeichen/s) und eine weitere Magnetbandeinheit. Die Anlage kann mit einem BASIC- (und/oder APL-)Compiler ausgerüstet werden. Den Benutzern stehen über 100 BASIC-Anwendungsprogramme aus dem Gebiet von Mathematik, Statistik und Betriebswirtschaftslehre zur Verfügung.[23] Verwiesen sei auch auf die IBM Systeme 5110 und /34.[23a]

mier- und Bedienungshandbuch. Fort Colans 1980. – Vgl. auch: Hewlett-Pakkard. BASIC/1000L Real-Time BASIC. Reference Manual. Cupertino 1980.

19 H. Knittel. Datenverarbeitung in der Feinwerktechnik. „CISAB" Information. Mitteilungen des Hewlett-Packard-Benutzerclubs: Computer in Schule und Ausbildung 1 (1973) 1, S. 6.

20 Huntington II Project, State University of New York. Hewlett-Packard Educational Users Group Newsletter 4 (1973) 3, December, p. 67.

21 George S. Day, Gerald J. Eskin, David B. Montgomery and Charles B. Weinberg. Cases in Computer and Model Assisted Marketing: Planning. Cupertino 1973; Gerald J. Eskin and David B. Montgomery. Cases in Computer and Model Assisted Marketing: Data Analysis. Cupertino 1975.

22 Hewlett-Packard. Time-Shared BASIC Program Library Handbook. Cupertino 1971; Hewlett-Packard. Time-Shared BASIC Handbook. November 1971 Supplement. Cupertino 1971.

23 IBM. IBM 5100. BASIC Introduction. 2nd ed., White Plains 1975; IBM. IBM 5100. BASIC Reference Manual. 4th ed., White Plains 1977; IBM. BASIC Benutzerhandbuch IBM 5100. Stuttgart 1976; IBM. IBM 5100 Stat/BASIC User's Guide. Atlanta 1975.

23a IBM. IBM 5110 BASIC Introduction. White Plains 1978; IBM. IBM 5110 BASIC Reference Manual. 3rd ed., White Plains 1979; IBM. IBM 5110. BASIC User's Guide. 3rd ed., White Plains 1979; IBM. IBM System/34. BASIC Handbuch. 3. Aufl., Stuttgart 1981.

122.6 Prime Computer, Inc.

Bei den Systemen Prime 100 und 200 ist BASIC als Programmiersprache für den Monologbetrieb (batch mode) und bei Prime 300 zusätzlich für den Dialogbetrieb (conversational mode) verfügbar. Als Terminals sind generell Datenstationen mit Druckausgabe resp. Sichtanzeige verwendbar; bei Stapelverarbeitung können zusätzlich Lochkarten-, Lochstreifen- oder Magnetbandeinheiten sowie Magnetplatteneinheiten (ASCII-Code) eingesetzt werden. Verfügbar ist eine BASIC-Grundversion ohne MAT und PRINT USING-Statements (bei einer minimalen Kernspeichergrösse von 16 K Worten). Ausgeweitete BASIC-Versionen (BASIC extended) sind verfügbar mit MAT-Befehlen oder mit MAT- und PRINT USING-Befehlen (bei minimaler Kernspeichergrösse von 32 K).[24]
Die Modelle Prime 100 und 200 sind bis auf 64 K ausbaubar; Prime 300 kann bis maximal 256 K Worte ausgebaut werden und ist damit eindeutig nicht mehr als Kleincomputer anzusprechen.[25]

122.7 Heinrich Dietz

Das Computersystem Dietz 621 lässt als Programmiersprache auch BASIC zu, wobei neben einer einfachen BASIC-Grundversion auch die Varianten C-BASIC und BASEX verfügbar sind.[26]
C-BASIC (Commercial BASIC) ist durch COBOL-orientierte Elemente (X-Zahl) gekennzeichnet; BASEX wurde am Institut für Physik der Universität Freiburg (Breisgau) entwickelt und dient insbesondere zur Prozesslenkung und Prozessdaten-Erfassung.

122.8 Nixdorf Computer AG

Für das System Nixdorf 8870/1 steht Business-Basic zur Verfügung. Es handelt sich hierbei um einen leistungsfähigen BASIC-Dialekt, der teilweise über die nach ANSI BASIC vorgesehenen Standards hinausgeht.[27]

24 Prime Computer. BASIC Interpretive Language. Natick 1974, p. 5.1
25 Prime Computer. Der Prime Computer Plan. Wiesbaden 1974.
26 Dietz. Dietz 621 Handbuch. Mülheim 1974, S. 235–254; Dietz. Dietz 621 BASEX Beschreibung, Mülheim 1975.
A. Goldenberg, Ch. Schlier und W. Schupp. Beschreibung der Prozesssprache BASEX. Freiburg 1974.
27 Nixdorf. Systemliteratur Nixdorf 8870/1. System-Software Business-Basic. Paderborn 1979.

Umfangreiche Anwendersoftware und grosse Teile des Bedienungssystems basieren auf BASIC-Programmen. Für die grösseren Systeme 8870/3 und /5 wird mit der Systemsoftware Version NIROS 4.1 ein sogenanntes „Structured BASIC" angeboten, das zu den Vorgängerversionen eine geringe Kompatibilität aufweist. Auf GOTO-Statements wird verzichtet und es werden nur blockstrukturierte Programme bearbeitet; bemerkenswert ist auch, dass die Variablennamen bis zu 28 Zeichen umfassen dürfen.[27a]

122.9 Ing. C. Olivetti & C. S. p. A.

Das Kleincomputersystem P 6060 lässt – ähnlich wie die Systeme Wang 2200 und IBM 5100 – die Direkteingabe der zum Aufbau von BASIC-Statements benötigten Kennworte (REM, DIM, DEF, FN, MAT, INPUT, READ, DATA, PRINT, USING, WRITE, ON, GOTO, GOSUB, RETURN, STOP, IF, THEN, FOR, TO, STEP, NEXT, END) über eine BASIC/Alphatastatur zu. Das System P 6060 bietet auch die Möglichkeit zur Definition von sequentiell und index-sequentiell aufgebautes Files, zur Verkettung von Programmen unter Verwendung eines COMMON-Bereichs, etc. Die verwendete BASIC-Version ist auf ECMA/ANSI Minimal BASIC abgestimmt.[28] Speziell zu erwähnen ist auch das BCS (Business Computer System) 2000.[28a]

27a Nixdorf. Systemliteratur 8870. Structured BASIC. Paderborn 1981.

28 Olivetti. BASIC Language XP 6000. Preliminary Copy. Ivrea 1976; Olivetti. P 6060 Reference Manual. Preliminary Edition. Ivrea 1976; Olivetti. P 6060 Anwenderhandbuch. [Frankfurt] 1976.

28a Olivetti. BCS 2000. BASIC Language Reference Manual. Ivrea 1981; Olivetti. BCS 2000. Business Computer Operating System (BCOS). BASIC Environment. Programmer Manual. Ivrea 1981.

Prozessor →	Computer Automation ALPH LSI	Datapoint	Dietz MINCAL 621 BASEX	Nixdorf 8870/1	Digital Equipment BASIC PLUS	PRIME Computer	WANG 2200
Elementaranweisungen							
REM	+	+	+	+	+	+	+
REMARK			+		+	+	
DIM	+	+	+	+	+	+	+
DEF numerisch	+		+	+	+	+	+
DEF alphanumer.	+			+	+		
DATA	+	+	+	+	+	+	+
data	+	+	+	+	+	+	+
READ	+	+	+	+	+	+	+
INPUT	+	+	+	+	+	+	+
RESTORE	+		+	+	+	+	+
PRINT	+	+	+	+	+	+	+
PRINT TAB	+			+	+		+
PRINT USING				+		+	+
IMAGE				+		+	+

Tabelle 1. Minicomputer – Verfügbarkeit von BASIC-Statements

Prozessor →	Computer Automation ALPH LSI	Datapoint	Dietz MINCAL 621 BASEX	Nixdorf 8870/1	Digital Equipment BASIC PLUS	PRIME Computer	WANG 2200
PAGE					+		
LET	+	+	+	+	+	+	+
let		+		+		+	+
GOTO	+	+	+	+	+	+	+
GOTO/ON	+		+	+	+	+	+
IF	+	+	+	+	+	+	+
FOR/NEXT	+	+	+	+	+	+	+
STOP	+		+	+	+	+	+
END	+		+	+	+	+	+
GOSUB	+	+	+	+	+	+	+
GOSUB/ON			+	+	+		+
RETURN	+	+	+	+	+	+	+
Matrizenanweisungen							
MAT READ	+			+	+	+	+
MAT INPUT	+			+	+	+	
MAT LET	+				+		
MAT let				+		+	+
MAT PRINT	+			+	+	+	+
MAT PRINT USING							

Tabelle 1 (Fortsetzung). Minicomputer – Verfügbarkeit von BASIC-Statements

Prozessor →	Computer Automation ALPH LSI	Datapoint	Dietz MINCAL 621 BASEX	Nixdorf 8870/1	Digital Equipment BASIC PLUS	PRIME Computer	WANG 2200
Fileanweisungen[1]							
OPEN		×	×	×	×	×	×
CLOSE			×	×			×
PUT/WRITE		×	×	×	×	×	×
MAT PUT/MAT WRITE				×		×	×
RESET					×	×	
GET/READ		×	×	×	×	×	×
MAT GET/MAT READ				×	×	×	×
Dateiorganisation							
sequentiell		×	×	×	×	×	×
random				×	×	×	×
index-sequentiell		×		×			×
Dateiart							
binär			×		×	×	×
BCD					×		
ASCII		×	×	×	×	×	
Diagnoseanweisungen							
PRINT ALL		×		×	×	×	×
DUMP		×			×		

[1] Die Statementangabe in der ersten Spalte bezieht sich auf den FOSBIC-Compiler; bei den mit × bezeichneten BASIC-Compilern sind Statements mit ähnlichen Funktionen aber anderslautenden Kennworten oder unterschiedlichem Statementaufbau verfügbar.

Tabelle 1 (Fortsetzung). Minicomputer – Verfügbarkeit von BASIC-Statements

123 Realisierter BASIC-Sprachumfang

Ausser den vorerwähnten Maschinentypen verfügen alle marktgängigen Kleincomputer über BASIC-Compiler. Einzelne von ihnen werden in der Tabelle 1 dargestellt, wobei die Unterschiede bezüglich des realisierten Sprachumfangs besonders zu beachten sind. Ähnliche Unterschiede lassen sich auch bezüglich der Minicomputer der Firmen

Cincinnati Milacron Inc.,
Computer Automation, Inc.,
Datapoint Corporation,
General Automation Inc.,
Harris Corporation,
Modular Computer Systems Inc.,
Perkin-Elmer und
Westinghouse Electronic Co.

feststellen. Einzelheiten sind den im Literaturverzeichnis (Band 3, Abs. 651) aufgeführten BASIC-Manualen zu entnehmen;[29] dort sind auch die für den Leser relevanten Kontaktadressen angeführt.

13 BASIC-Hochschuleinsatz

131 Allgemeines

BASIC wird in den USA insbesondere auch an Business Schools als Programmiersprache verwendet. Im Rahmen einer Umfrage bei den 147 Mitgliedern der American Association of Collegiate Schools of Business (AACSB)[30] konnte für 1974 festgestellt werden, dass Programmierkenntnisse fast durchwegs als Voraussetzung für den Studienabschluss

29 Vgl. auch: Karl Weber. BASIC Programmier-Richtlinien für Austauschprogramme. Bern 1979, S. 49–81.

30 Über die AACSB vgl. Hanns-Martin Schönfeld. Betriebswirtschaftslehre im anglo-amerikanischen Raum. In: Erwin Grochla und Waldemar Wittmann. Handwörterbuch der Betriebswirtschaft. 4. Aufl., Bd. 1, Stuttgart 1974, Sp. 755; Karl Weber. Die Lehre des Faches Management Science an den anglo-amerikanischen Business Schools. In: Gert v. Kortzfleisch. Wissenschaftsprogramm und Ausbildungsziele der Betriebswirtschaftslehre. Berlin 1971, S. 243–245.

verlangt werden.[31] Wie aus Tabelle 2 hervorgeht, wurden (bei 107 sich an der Umfrage beteiligenden Schulen) BASIC-Programmierkenntnisse im Rahmen des Grundstudiums (bachelors level) in 19% aller Fälle und im Rahmen des Aufbaustudiums in 28% (masters level) resp. 4% (doctorate level) aller Fälle verlangt.[32]

BASIC-Programmierkenntnisse gelten an amerikanischen Hochschulen vielfach als Voraussetzung für den Studienabschluss.

Höhere Verbreitungsziffern liessen sich im Rahmen der vorerwähnten Untersuchung – wie bereits 1970 – nur noch für FORTRAN feststellen. Mit zunehmender Verfügbarkeit von BASIC-Anwendersoftware dürfte BASIC, jedenfalls in den USA, bald zur gebräuchlichsten Programmier-

Sprache	Programmierkenntnisse erforderlich für		
	Undergraduate Studies	Graduate Studies	Postgraduate Studies
FORTRAN	48%	81%	39%
BASIC	19%	28%	4%
COBOL	5%	14%	4%
PL/1	6%	3%	4%
APL	1%	6%	4%
Übrige (RPG, Assembler, etc.)	2%	8%	4%
Zahl der Anstalten	81	58	46

Quelle: Report on the 3rd Triennial Survey of Computer Uses and Computer Curriculum in Schools of Business. Computing Newsletter for Schools of Business 8 (1974) 2, October, p. 2

Tabelle 2. Programmiersprachen an amerikanischen Business Schools

31 "In 1970, 62 percent of schools required a programming proficiency at the bachelors level. That level has grown to 72 percent. The more dynamic impact is at the graduate level. Only 29% of schools required programming at the doctorate level in 1970; the 1974 level was 44%. Only 55% of schools required programming at the masters level in 1970; the 1974 level was 78%." Report on the 3rd Triennial Survey of Computer Uses and Computer Curriculum in Schools of Business. Computing Newsletter for Schools of Business 8 (1974) 2, October, p. 2.

32 Für die Umfrageergebnisse von 1970 vgl. Updating the Survey on Computer Uses and Computer Curriculum. Computing Newsletter for Schools of Business 4 (1970) 2, October, p. 1 und 4; Weber, Die Lehre des Faches Management Science an den anglo-amerikanischen Business Schools, S. 250.

sprache für Wirtschaftswissenschaftler werden.[33] An manchen Hochschulen vermochte BASIC die Programmiersprache FORTRAN bereits weitgehend zu verdrängen.[34]
In der Bundesrepublik Deutschland und in der Schweiz vermochte sich BASIC bisher im Hochschulbetrieb noch nicht in gleichem Masse durchzusetzen, wie in den USA. Erwähnt sei immerhin, dass an der Justus-Liebig-Universität Giessen BASIC (neben FORTRAN und COBOL) bereits 1970 in das Grundstudium für Wirtschaftswissenschaftler integriert wurde. In neuerer Zeit wird BASIC auch an anderen Hochschulen (Essen, Dortmund) als Programmiersprache für Wirtschaftswissenschaftler verwendet. Vgl hierzu auch Tabelle 3.

Sprache	Studiengänge	Sprache	Studiengänge
COBOL	12	BASIC	3
FORTRAN	10	ALGOL	2
PL/I	6	Assembler	2
PASCAL	3	APL	1

Quelle: H. Oesterle. Betriebsinformatik im Grundstudium wirtschaftswissenschaftlicher Studiengänge. Angewandte Informatik 23 (1981) 1, S. 29.

Tabelle 3. Programmiersprachen in wirtschaftswissenschaftlichen Studiengängen der Bundesrepublik Deutschland

33 Die in Tabelle 2 dargestellten Umfrageergebnisse wurden von Daniel Couger wie folgt kommentiert: "The researcher expected BASIC to gain inroads on FORTRAN as the required language. Such was not the case... Two reasons exist: (1) many library programs are written in FORTRAN – instructors like to have students capable of revising these programs. (2) FORTRAN is more of a universal language – more changes are required in transferring BASIC programs." Report on the 3rd Triennial Survey of Computer Uses and Computer Curriculum in Schools of Business, p. 2.

34 Don Cassel. BASIC Programming in Real Time. Reston 1975, p. V: "At Humber College we have replaced Fortran with Basic for Business students. We have found it capable of demonstrating the majority of introductory problems in programming. It can also be used successfully as a foundation for other more complex languages."

Mit zunehmender Verbreitung der Programmiersprache BASIC an den amerikanischen Hochschulen wurde ein grosser Bedarf an Anwendersoftware deutlich, der – soweit nicht auf Programmbibliotheken der Computerhersteller zurückgegriffen werden konnte – durch Eigenentwicklungen gedeckt werden musste. Pionierarbeit wurde wiederum am Dartmouth College mit Schaffung der DTSS Software geleistet; verfügbar sind etwa 500 Programme aus dreissig verschiedenen Anwendungsbereichen.[35] Speziell auf Probleme der betriebswirtschaftlichen Rechnungsführung ausgerichtet ist die Programmsammlung von G. Cook et al.[36] BASIC-Programme aus dem Gebiet des Operations Research finden sich im Lehrbuch von A. O. Converse[37] und bei C. McMillan und R. Gonzales.[38] Auch Forschungsberichte und Arbeitspapiere amerikanischer Hochschulinstitute enthalten oft beachtenswerte BASIC-Programme.[39] Der grosse Bedarf an BASIC-Programmen führte an amerikanischen Hochschulen – speziell für das Gebiet der Betriebswirtschaftslehre – zur Errichtung von Programmaustauschstellen.

BASIC-Programme von allgemeinem Interesse werden von BASIC User Groups gesammelt und weitervermittelt.

Die Professur für Betriebswirtschaftslehre V der Justus-Liebig-Universität Giessen hat sich zur Aufgabe gestellt, für den deutschen Sprachbereich – in Zusammenarbeit mit amerikanischen Hochschulen – als Vermittlungsstelle für BASIC-Programme zu dienen. Eine umfangreiche BASIC-Programmsammlung soll als Ergänzung zum vorliegenden Lehr- und Handbuch veröffentlicht werden.

35 Honeywell. Series 600/6000 Software. Introduction to DTSS Software. Waltham 1973. – DTSS ist ein Lizenzprodukt der (mit dem Dartmouth College verbundenen) Dartmouth Time-Sharing System, Inc. Vgl. auch Bd. 1, Abs. 112.2.

36 Gregory A. Cook, Barbara J. Wade and Clark C. Upton. Computer Accounting Methods Featuring BASIC Programming on a Time-sharing Computer. New York 1974.

37 A. O. Converse. Optimization. New York 1970.

38 Claude McMillan and Richard F. Gonzales. Systems Analysis. A Computer Approach to Decisior Models. 3rd ed., Homewood 1973.

39 Vgl. etwa: Elbert B. Greynolds, Jr. Time Sharing: Computer Programs and Applications in Accounting. Atlanta 1974; Eliezer Naddor. Inventory Simulations im Time-Sharing and Some Programming and Documentation Considerations. Baltimore 1974.

133 FOSBIC-Compilersystem

Bei Veröffentlichung der ersten Auflage dieser Arbeit wurde festgehalten, dass die Hochschulrechenzentren in Deutschland Mitte der 70er Jahre noch zumeist im Batchbetrieb arbeiteten.[40] Diese Situation hat sich im Laufe der vergangenen Jahre insofern wesentlich verändert, als nunmehr an den meisten Hochschulen extensive Möglichkeiten zum Dialogverkehr bestehen.

Die genannte Entwicklung beeinflusste auch den Ausbau des FOSBIC-Compilersystems wesentlich. War die FOSBIC Version 1 noch (fast) ausschliesslich auf den Batchbetrieb ausgerichtet[41], so ist die Version 2 nunmehr in gleichem Masse für den Monolog- und Dialogbetrieb geeignet. Einzelheiten zur Entwicklung und Verbreitung des FOSBIC-Compilersystems sind aus Band 1 ersichtlich.[42]

Bei grösserer Verfügbarkeit von BASIC-Prozessoren dürfte sich in Zukunft auch der BASIC-Einsatz an den deutschen Hochschulen in vermehrtem Masse den vorgezeichneten amerikanischen Entwicklungen anpassen.[43] Dazu vermag das versatile – auf FORTRAN IV basierende – FORTRAN Simulated BASIC Interpretive Compiler-Compilersystem (FOSBIC) sicherlich auch weiterhin beizutragen. Vgl. Vorwort, S. 5. Vor allem können mit dem FOSBIC-Compilersystem leicht BASIC-Spracherweiterungen auf ihre Benutzerfreundlichkeit getestet werden, ohne dass dabei umfangreiche Compilerentwicklungsarbeiten durchgeführt werden müssen. Bezüglich der Kennwortsyntax ist die FOSBIC Version 2 so flexibel, dass innerhalb einer Dialogsitzung mehrere Kennwortsysteme eingegeben und getestet werden können. Ein Eingriff in die Programmstruktur des FOSBIC-Compilersystems ist dazu nicht notwendig. Diese Flexibilität erleichtert insbesondere auch den Übergang von ANSI BASIC 79 zu ANSI BASIC 82, da diese beiden Versionen teilweise – beispielsweise bezüglich der Fileanweisungen – unterschiedliche Kennwortstrukturen aufweisen.

40 Karl Weber und Carl Wolfram Türschmann. BASIC 2. 1. Aufl., Bern 1977, S. 30.

41 Vgl. Weber und Türschmann, FOSBIC-Compiler, S. 91–94.

42 Vgl. Bd. 1, S. 30–32, 38–42, 5–6.

43 Vgl. Tabelle 2 und 3.

14 BASIC-Standardisierung

Die mit der Standardisierung von Minimal BASIC verbundenen Arbeiten konnten in den USA Ende 1975 im wesentlichen abgeschlossen werden.[44]

Damit lief die Tätigkeit der ANSI BASIC-Kommission X3J2 aber nicht aus. Auf Anregung von Gerald Hornik (Digital Equipment Corporation) und mit Unterstützung der National Bureau of Standards Task Group 16 on BASIC[45] wurde vielmehr die Bildung mehrerer Enhancement BASIC Subcommittees beschlossen und ihnen die Problemkreise

- Grundsprachumfang (nucleus),
- Matrizen (matrices),
- Dateien (files),
- Zeichenketten (strings),
- Formatierung (formatting),
- Programmverkettung (inter-program-linkage)

zur Bearbeitung zugewiesen.[46] Auf die Bildung weiterer – teilweise ebenfalls angeregter – Unterkommissionen (error control, debugging, intelligent terminal handling, graphics, real-time/process control) wurde einstweilen verzichtet.[47]

Die Unterkommissionen hatten zunächst funktional gegliederte Prioritätstabellen zu erstellen, aus denen die erwünschten BASIC-Erweiterungen – grob gegliedert – ersichtlich waren. Als Ausgangsbasis diente eine

44 Vgl. Bd. 1, Abs. 14.

45 X3J2/75-07, United States of America, General Services Administration. Proposal for Work on Enhancements. – Vgl. auch Abs. 331 und Abs. 332.

46 Mitglieder der Unterkommissionen: Nucleus: Dartmouth College (S. J. Garland), University of Iowa (G. L. Isaacs), Digital Equipment Corporation (G. J. Hornik) u. a. – Matrices: Dartmouth College (T. E. Kurtz) u. a. – Files: Virginia Polytechnic Institute (J. A. Lee), Digital Equipment Corporation (G. J. Hornik) u. a. – Strings: Hatfield Polytechnic (G. M. Bull) u. a. – Formatting: Dartmouth College (T. E. Kurtz); University of Iowa (G. L. Isaacs) u. a. – Inter-Program Linkage: Dartmouth College (S. J. Garland), Hatfield Polytechnic (G. M. Bull) u. a.

47 X3J2/75-23, B-3. – Die mit der BASIC-Standardisierung verbundenen Schwierigkeiten dürfen keineswegs unterschätzt werden, worauf M. F. Lipp schon 1969 hinwies: "In a few years, the cry for a standard definition [of BASIC] may arise. But it is doubtful if much beyond [what is now called Minimal BASIC] will ever be standardized. And so we will recognize a BASIC family: BASIC will begett BASIC – unto generation after generation." Michael F. Lipp. The Language BASIC and its Role in Time Sharing. Computers and Automation 18 (1969) October, p. 42–43.

von der NBS Task Group 16 erstellte Liste[48], über deren Inhalt Tabelle 4 orientiert. Aus dieser Tabelle sind auch die allgemeinen BASIC-Entwicklungstendenzen deutlich erkennbar.
Die Prioritätenliste der NBS Task Group 16 wurde von allen Kommissionen – insbesondere unter Verwendung der einschlägigen Kapitel aus NCC Standard BASIC[49] – überarbeitet.[50]
Die Resultate der Kommissionsarbeit wurden 1979 bereinigt und im
– Preliminary Draft, American National Standard for BASIC
zusammengefasst.[51] Besonders wichtig erscheinen die Standardisierungsvorschläge für Matrizen- und Fileanweisungen. Sie wurden in die Abschnitte 321 und 322 des vorliegenden Buches eingearbeitet. Ebenso bilden sie die Grundlage der in Abschnitt 333 dargestellten erweiterten Programmsteuerungsanweisungen sowie der in Abschnitt 334 (Bannd 3) behandelten Graphikanweisungen.
1982 wurde ein – in wesentlichen Teilen überarbeiteter – Standardisierungsvorschlag als
– Draft Proposed, American Standard for BASIC
veröffentlicht.[52]

48 X3J2/75-07, p. 2–10.
49 G. M. Bull, W. Freeman and S. J. Garland. Specification for Standard BASIC. Manchester 1973.
50 Zwischen X3J2-Dokumenten und NCC Standard BASIC bestehen u. a. folgende Entsprechungen: X3J2/75-17, Nucleus – functions definitions = NCC Standard BASIC, p. 70–76; X3J2/75-18, Nucleus and Strings – standard functions = NCC Standard BASIC, p. 57–69; X3J2/75-15, Files = NCC Standard BASIC, p. 152–157; X3J2/75-16, Files – Input and Linput = NCC Standard BASIC, p. 101–109; X3J2/75-13, Files and Formatting – Print Statement = NCC Standard BASIC, p. 89–100; X3J2/75-12, Inter-program linkage – chaining = NCC Standard BASIC, p. 158–161; X3J2/75-14, Inter-program linkage – subprograms and libraries = NCC Standard BASIC, p. 162–174.
51 ANSI, American National Standards Institute. American National Standard for BASIC. Preliminary Draft. September 1979. Hanover 1979. – Im folgenden zitiert als ANSI BASIC.
52 ANSI, American National Standards Institute. American National Standard for BASIC. Draft Proposed, February 15, 1982. Hanover 1982. – Im folgenden zitiert als ANSI BASIC 82.

Grundsprachumfang (nucleus)	
I	Mehrfaches LET, wie LET A = B = C = 100 IF/THEN GOSUB, wie IF A = 1 THEN GOSUB 100 ON/GOTO, wie ON A GOTO (100, 200, 300) Stringnamen: A$, A0$, A1$, . . ., Z8$, Z9$ Funktionen: CLG, Logarithmus zur Basis 10; DAT$, Datum; CLK$, Uhrzeit; MAX, Maximalwert; MIN, Minimalwert; MOD, Modus; VAL, numerisch umgesetzter Stringwert; STR$, alphanumerisch umgesetzter Ziffernwert.
II	IF/ THEN ⟨Statement⟩, wie IF Y = 10 THEN PRINT „Y = 10" Mehrzeilige DEF-Funktionen, mit DEF FN und FNEND Kleinbuchstaben, im ASCII-Code Funktionen: TIM, Programmausführungszeit in Sekunden; ARCCOS, ARCSIN, zyklometrische Funktionen; HYPSIN, HYPTAN, HYPCOT, hyperbolische Funktionen; MAT INPUT, MAT OUTPUT, Matrizeneingabe und -ausgabe.
Matrizen und Verteilungsfunktionen (matrices and other math)	
I	MAT-Statements vom Typ MAT A = IND, Einheitsmatrix; MAT A = ZER, Nullmatrix; MAT A = ONE, Einermatrix; MAT A = −A; MAT A = A + B; MAT A = INV, Kehrmatrix, mit DET; MAT A = TRN, transportierte Matrix; MAT A = A ( ), Redimensionierung.
II	Dreiecksmatrizen; Eigenwerte von Matrizen; Matrixspur. Verteilungsfunktionen: Hyperbolische, Gamma-, Normal-Verteilung.
Dateien (files)	
I	Sequentielle Dateien (symbolic, sequential files) IF END; INPUT; PRINT; RESTORE; SCRATCH; APPEND; CLOSE.
II	Direktzugriffs-Dateien (internal, random access files). Scratch Files. LINPUT; MARGIN; DELIMIT.
III	Indexsequentielle Dateien (indexed sequential symbolic files) POSITION; UPDATE.
Formatierung (formatting)	
I	PRINT USING Datendarstellung: Links-, rechtsbündig und zentrierte Stringausgabe; Ziffern mit gleitender Einfügung des Währungszeichens ($); führende Sternzeichen ($***5.00); Saldovermerk (DB, CR)

Tabelle 4. Enhancement BASIC, NBC Task Group 16. Spracherweiterungsmöglichkeiten.

Fortsetzung von Tabelle 4.

Zeichenketten (strings)	
I	Vektorielle Stringdarstellung, wie DIM A$ (6) Funktionen: LEN, Zeichenzahl; TRM$, Abschneiden nachfolgender Leerzeichen; SUB$, Substring eines Strings; POS, Position des Substrings innerhalb eines Strings. Verschachtelte Stringfunktionen, wie LEN(SUB$) ... LINPUT, im Dialogverkehr zur zeilenweisen Eingabe von Strings.
II	Stringverkoppelung, wie LET A$ = B$ + C$ Funktionen: UP$, Grossbuchstabenwert; LOC$, Kleinbuchstabenwert. DEF-Funktionen, wie DEF FNA$ (X$) = ...
Programmverkettung (interprogram linkage)	
I	CHAIN
II	CALL ⟨Subroutine⟩ LIBRARY ⟨Bibliothek-Subroutine⟩ CALL ⟨Bibliothek-Subroutine⟩
III	Overlay-Technik (overlay capability)
Programmkontrolle/-Bereinigung (control/debug)	
I	BREAK ON; TRACE ON; STEP
II	ON ATTENTION; IF BREAK; ON ERROR; ON OVERFLOW; ON UNDERFLOW ELS, Nummer des fehlerhaften Statements; EST, Nummer des Statementart; ESM, Nummer der Fehlermeldung; ASL, Nummer des angemahnten Statements.
III	ON ENDPAGE; ON ENDFILE

Von der ECMA BASIC-Kommission TC 21 wurde 1975 ebenfalls mit der Arbeit an Enhancement BASIC begonnen.[53] Die Untersuchungen liefen im wesentlichen mit denen der ANSI Enhancement Subkommission parallel und wurden ständig aufeinander abgestimmt.
Die vorliegende Arbeit nimmt vorwiegend auf ANSI BASIC 79 Bezug, verdeutlicht aber auch wesentliche Unterschiede zwischen ANSI BASIC 79 und ANSI BASIC 82.

53 Es werden sieben Modelle bearbeitet: Nucleus (Philips), Matrices (Siemens), Files (Honeywell-Bull), Strings (Nixdorf), Formatting (IBM), IPL (ICL) und Graphics (Olivetti). – Bericht an X3J2 Mitglieder. X3J2/75-33. C.

2 BASIC-Problembearbeitung

21 Problembeschreibung und Problemanalyse

Innerhalb der Matrizenrechnung nimmt die Matrizeninversion eine zentrale Stellung ein. Da eine entsprechende MAT-Funktion[1] zur vereinfachten Lösung dieses Problems nicht von allen BASIC-Compilern zur Verfügung gestellt wird, soll hier im Rahmen der BASIC-Problembearbeitung eine Beispiellösung vorgestellt werden.
Existiert zu einer Matrix D eine Matrix X, so dass deren Matrizenprodukt

$$E = D \cdot X$$

zu einer Einheitsmatrix E führt, so bezeichnet man die Matrix X als die Inverse oder Kehrmatrix der Matrix D und kennzeichnet dies durch eine hochgestellte -1[2,3].

$$X = D^{-1}$$

Voraussetzung für die Existenz der Kehrmatrix ist, dass die Determinante der gegebenen Ausgangsmatrix D ungleich Null ist, d. h. die Matrix D darf nicht singulär sein.
Die inverse Matrix bestimmt sich aus der Lösung von n Gleichungssystemen der Art

$$\begin{aligned} Dx_1 &= e_1 \\ Dx_2 &= e_2 \\ &\vdots \\ Dx_n &= e_n \end{aligned}$$

wobei e_i die n Einheitsvektoren der Einheitsmatrix E und die x_i die n Spaltenvektoren der gesuchten Kehrmatrix darstellen. Für die m-te Spalte der Inversen ergibt sich folgendes Gleichungssystem:

1 Vgl. Abs. 331.221.
2 Heinz Stöwe und Erich Härtter. Lehrbuch der Mathematik für Volks- und Betriebswirte. 2. Aufl., Göttingen 1972, S. 181.
3 Heiner Müller-Merbach. Operations Research. Methoden und Modelle der Optimalplanung. 3. Aufl., München 1973, S. 211.

$$
\begin{array}{l}
d_{11}x_{1m} + d_{12}x_{2m} + \ldots + d_{1m}x_{mm} + \ldots + d_{1n}x_{nm} = 0 \\
d_{21}x_{1m} + d_{22}x_{2m} + \ldots + d_{2m}x_{mm} + \ldots + d_{2n}x_{nm} = 0 \\
\quad\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
d_{m1}x_{1m} + d_{m2}x_{2m} + \ldots + d_{mm}x_{mm} + \ldots + d_{mn}x_{nm} = 1 \\
\quad\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
d_{n1}x_{1m} + d_{n2}x_{2m} + \ldots + d_{nm}x_{mm} + \ldots + d_{nn}x_{nm} = 0
\end{array}
$$

Zur Lösung derartiger Gleichungssysteme werden eine Reihe von Methoden angeboten[4], wobei die Eliminationsverfahren gegenüber den Iterationsverfahren aufgrund eines wesentlich geringeren Rechenaufwandes bevorzugt werden[5].

Als Bewertungsgrundlage für die Alternativenwahl können hier drei Ziele angegeben werden[6]:

- Programmierbarkeit des Algorithmus,
- Konvergenz und numerische Stabilität sowie
- allgemeine und variable Anwendbarkeit.

Unter der Programmierbarkeit wird vorab die Übersetzbarkeit des Algorithmus in ein Programm verstanden; hardwarebezogene Grössen wie Kernspeicherbedarf, Statementumfang und Rechengeschwindigkeit sind mitzuberücksichtigen. Die Konvergenz des Verfahrens soll eine Lösung auch beim Vorliegen spezieller Datenstrukturen sicherstellen. Die numerische Stabilität bezieht sich vor allem auf die erzielbare Rechengenauigkeit auch unter ungünstigen Ausgangsdaten. Allgemeine und variable Anwendbarkeit findet sich in zusätzlichen Ergebnissen, die der Algorithmus liefern kann. Wichtig in diesem Zusammenhang wäre auch die Lösung linearer Gleichungssysteme, Berechnung der Determinante oder auch die Anpassung an spezielle Matrizenstrukturen.

Eine umfassende Diskussion der verschiedenen Inversionsverfahren findet sich bei Werner Dück[7] und sollte vom Anwender bei einer genaueren Problemanalyse herangezogen werden.

22 Programmplanung

Eine einfach zu programmierende, konvergente, allgemein anwendbare

4 Werner Dück. Numerische Methoden der Wirtschaftsmathematik I. Berlin 1970, S. 240–261, S. 305–306, S. 314–315 und S. 323–340.

5 Dück, Numerische Methoden der Wirtschaftsmathematik I, S. 254.

6 Dück, Numerische Methoden der Wirtschaftsmathematik I, S. 261–264 und S. 340–345.

7 Dück, Numerische Methoden der Wirtschaftsmathematik I, S. 342–344.

und im Lösungsweg übersichtliche Methode stellt das Austauschverfahren bei vorgegebener Pivotwahl dar. Es wird auch als abgewandeltes Gauss-Jordan-[8] oder Gauss-Jordan-Rutishauser-Verfahren[9] bezeichnet. Hierbei wird die Tatsache benutzt, dass die Berechnung der Koeffizienten der Ursprungsmatrix in den einzelnen Schritten der Austauschmethode unabhängig von der rechten Seite des Gleichungssystems ist und dass die Berechnung der Koeffizienten der rechten Seite – hier steht zu Beginn des Austauschverfahrens eine Einheitsmatrix – für alle n Einheitsvektoren einheitlich geschieht. Da die Wahl der Pivotelemente mit der Hauptdiagonalen der Ausgangsmatrix vorgegeben ist, muss stets gewährleistet sein, dass während der Berechnung das entsprechende Pivotelement ungleich Null ist. Andernfalls liegt eine zumindest für dieses Verfahren numerisch singuläre Matrix[10] vor.

Jeder Austauschschritt des Verfahrens gliedert sich in zwei Einzelschritte.

1. Schritt: Alle Elemente der beiden Matrizen werden in der Pivotzeile durch das Pivotelement dividiert. Für die m-te Zeile nach m Austauschschritten gilt:

	linke Seite	*rechte Seite*
	m-te Spalte	
	$0\ 0 \ldots 1 \ldots \frac{d_{mi}^{(m-1)}}{d_{mm}^{(m-1)}} \cdots \frac{d_{mn}^{(m-1)}}{d_{mm}^{(m-1)}}$	$\frac{e_{m1}^{(m-1)}}{d_{mm}^{(m-1)}} \cdots \frac{e_{mn}^{(m-1)}}{d_{mm}^{(m-1)}}$

Der hochgestellte Index bezeichnet die Elementwerte nach $(m-1)$ Austauschschritten, so dass gilt

$$d_{mi}^{(m)} = \frac{d_{mi}^{(m-1)}}{d_{mm}^{(m-1)}} \text{ und } e_{mi}^{(m)} = \frac{e_{mi}^{(m-1)}}{d_{mm}^{(m-1)}}.$$

2. Schritt: Von allen übrigen Zeilen j (j m) wird die Pivotzeile – multipliziert mit dem jeweiligen Zeilenelement aus der Pivotspalte – subtrahiert. Alle Elemente der Pivotspalte ausser dem Pivotelement selbst werden dabei auf den Wert Null gesetzt. Für die Zeile j im m-ten Austauschschritt gilt:

8 Claude McMillan and Richard F. Gonzales. Systems Analysis. A Computer Approach to Decision Models. 3rd ed., Homewood 1973, p. 409.

9 Dück, Numerische Methoden der Wirtschaftsmathematik I, S. 330–331.

10 Die Ursprungsmatrix braucht deshalb im mathematischen Sinn noch nicht singulär zu sein.

linke Seite

$$0\ 0 \ldots (d_{ji}^{(m-1)} - d_{jm}^{(m-1)} \frac{d_{mi}^{(m-1)}}{d_{mm}^{(m-1)}}) \ldots (d_{jn}^{(m-1)} - d_{jm}^{(m-1)} \frac{d_{mn}^{(m-1)}}{d_{mm}^{(m-1)}})$$

rechte Seite

$$(e_{j1}^{(m-1)} - e_{jm}^{(m-1)} \frac{e_{m1}^{(m-1)}}{d_{mm}^{(m-1)}}) \ldots\ldots\ldots\ldots (e_{jn}^{(m-1)} - e_{jm}^{(m-1)} \frac{e_{mn}^{(m-1)}}{d_{mm}^{(m-1)}})$$

Die Klammerausdrücke ergeben dabei die Elementwerte $d_{ji}^{(m)}$ bzw. $e_{ji}^{(m)}$ des m-ten Austauschschrittes.
Nach n Austauschschritten steht auf der linken Seite eine Einheitsmatrix und auf der rechten Seite die gesuchte Kehrmatrix.
Zu diesem Verfahren wird ein äquivalentes Flowchart entwickelt, wobei zunächst nur die Einzelschritte eines Austauschvorgangs berücksichtigt werden. Das Pivotelement d_{mm} muss dabei in einer Variablen T zwischengespeichert werden, da der Speicherplatz des Pivotelements während der Rechenoperation im ersten Einzelschritt auf den Wert 1 gesetzt wird. Beim zweiten Einzelschritt muss das jeweilige Zeilenelement d_{jm} aus der Pivotspalte m zwischengespeichert werden, da der Speicherwert dieses Elements während der Rechenoperation ebenfalls zerstört wird. Als Zwischenspeicher kann wiederum die Variable T dienen.

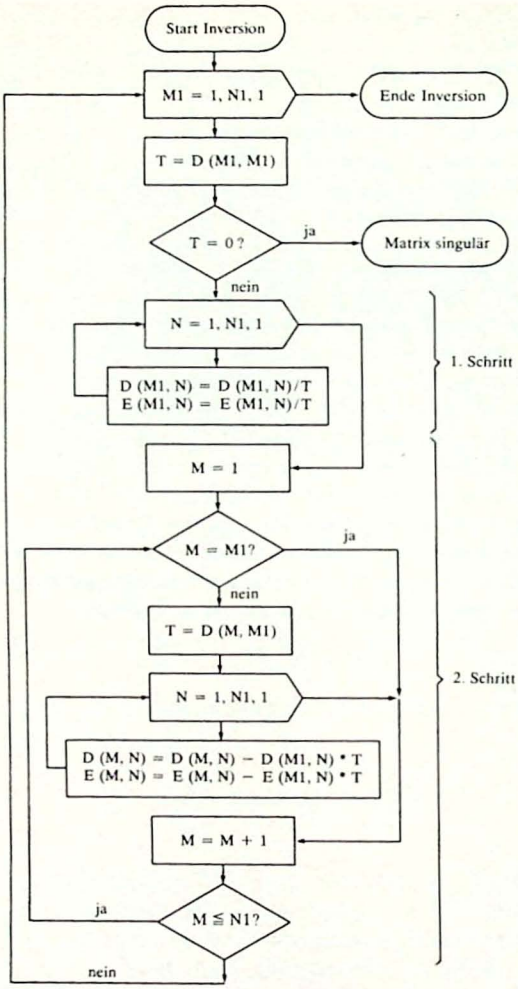


Abbildung 4.

Vor der Programmierung des obigen Flowchart sind noch die Eingabe der Ausgangsmatrix, die Bildung einer Einheitsmatrix, die Ausgabe der Kehrmatrix und eine eventuelle Kontrolle, die durch die Multiplikation der Ursprungsmatrix D mit der in E enthaltenen Kehrmatrix D^{-1} nach

$$P = D \cdot E$$

durchgeführt werden kann, hinzuzufügen. Die Ergänzung der Abbildung 3 für diese zusätzlichen Operationen sei dem Leser überlassen, wobei die Abbildungen 8 und 10 aus Band 1 eine wertvolle Hilfe sein können.

23 Programmierung

Um eine möglichst hohe Kompatibilität zu gewährleisten, sollte die Codierung des Flowcharts nach Abbildung 4 und aller Zusatzoperationen nur in BASIC-Elementaranweisungen erfolgen, wie das folgende BASIC-Programm zeigt.

```
  5 REM MATRIZENINVERSION
 10 DIM D(10,10),E(10,10),P(10,10)
 12 READ N1
 14 DATA 4
 16 PRINT #EINGABEMATRIX#
 18 FOR I=1 TO N1
 20   FOR J=1 TO N1
 22     READ D(I,J)
 24     PRINT D(I,J),
 26   NEXT J
 28   PRINT
 30 NEXT I
 31 REM BILDEN EINER EINHEITSMATRIX
 32 FOR I=1 TO N1
 34   FOR J=1 TO N1
 36     LET E(I,J)=0
 38     IF I <> J GOTO 42
 40     LET E(I,J)=1
 42   NEXT J
 44 NEXT I
 46 REM BEGINN DER INVERSION
 80 FOR M1=1 TO N1
 90   LET T=D(M1,M1)
100   IF T = 0 GOTO 360
105   REM BEGINN DES ERSTEN EINZELSCHRITTS
110   FOR N=1 TO N1
120     LET E(M1,N)=E(M1,N)/T
130     LET D(M1,N)=D(M1,N)/T
140   NEXT N
150   LET M=1
155   REM BEGINN DES ZWEITEN EINZELSCHRITTS
160   IF M1=M GOTO 240
170   LET T=D(M,M1)
180   FOR N=1 TO N1
210     LET E(M,N)=E(M,N)-E(M1,N)*T
220     LET D(M,N)=D(M,N)-D(M1,N)*T
230   NEXT N
240   LET M=M+1
250   IF M LE N1 GOTO 160
260 NEXT M1
265 REM ENDE DER INVERSION
270 PRINT
275 PRINT #INVERSE MATRIX#
280 FOR I=1 TO N1
282   FOR J=1 TO N1
284     PRINT E(I,J),
286   NEXT J
288   PRINT
290 NEXT I
292 RESTORE N1**2
294 FOR I=1 TO N1
296   FOR J=1 TO N1
298     READ D(I,J)
300   NEXT J
302 NEXT I
304 FOR I=1 TO N1
306   FOR J=1 TO N1
308     LET P(I,J)=0
310     FOR K=1 TO N1
312       LET P(I,J)=P(I,J)+D(I,K)*E(K,J)
314     NEXT K
316   NEXT J
318 NEXT I
319 PRINT
320 PRINT #KONTROLLMATRIX#
322 FOR I=1 TO N1
```

```
324   FOR J=1 TO M1
326     PRINT P(I,J),
328   NEXT J
330   PRINT
332 NEXT I
340 DATA 49.96, 12.23, 36.78, 29.43
342 DATA 22.46, 38.72, 40.15, 11.29
344 DATA 36.45, 19.20, 37.28, 6.43
346 DATA 17.84, 40.02, 27.86, 39.27
350 STOP
360 PRINT #EINGABEMATRIX IST SINGULAER#
400 END

*************** EVERYTHING SEEMS OK -- LET#S GO AHEAD

PERCENT OF AVAILABLE STORAGE USED                  42.622
PERCENT OF AVAILABLE DATA STORAGE USED              5.152
PERCENT OF AVAILABLE NUMBERED STATEMENTS USED      22.941
```

Die Anweisungsfolge von Statementnummer 80 bis 260 stellt die eigentliche Matrizeninversion dar. Mit der Abfrage

```
100 IF T = 0 GOTO 360
```

wird während des Austauschverfahrens auf Singularität abgeprüft und bei Vorliegen auf

```
360 PRINT #EINGABEMATRIX SINGULAER#
```

verzweigt. Durch die Statementfolge 32 bis 44 wird in dem Feld E eine Einheitsmatrix gebildet.
Wird das oben angegebene BASIC-Programm von dem verwendeten BASIC-Compiler ohne syntaktische Fehlermeldung umgewandelt, so kann mit dem Programmtest durch verschiedene Datenstrukturen fortgefahren werden. Hierzu kann die Kontrollmultiplikation der Ausgangsmatrix mit der Kehrmatrix – Statementfolge 304 bis 318 – wertvolle Hinweise auf die Rechengenauigkeit geben. Zuvor muss die Ausgangsmatrix nochmals eingelesen werden, da sie während der Inversionsrechnung zerstört wird.
Beim FOSBIC-Compiler zeigt sich durch die Kontrollrechnung, dass die Elemente der Kehrmatrix für das vorgegebene Datenbeispiel auf mindestens 10 Stellen genau berechnet werden.

```
EINGABEMATRIX
      49.96          12.23          36.78          29.43
      22.46          38.72          40.15          11.29
      36.45          19.2           37.28           6.43
      17.84          40.02          27.86          39.27

INVERSE MATRIX
      -.202807       -.327833       0.420606       0.177371
      -.213983       -.278288       0.377367       0.178582
      0.290547       0.443527       -.546334       -.255881
      0.104076       0.117875       -.188057      -.0556289

KONTROLLMATRIX
        1          -9.89530E-10    1.16415E-10   -2.91038E-11
  7.27596E-10          1          5.82077E-11   -5.09317E-10
  4.65661E-10      -4.94765E-10        1         1.38243E-10
  6.98492E-10      -2.32831E-10    2.32831E-10        1
```

Die maximale Genauigkeit des verwendeten Computersystems – CD 3300 – liegt bei 14 Stellen. Welche Rechengenauigkeit erreicht werden kann, hängt von der Struktur der Ausgangsmatrix und vom Rechenverfahren ab. Die numerisch kritische Stelle in dem hier angegebenen Austauschverfahren stellt die Umwandlung der ausserhalb der Pivotzeile und Pivotspalte stehenden Elemente im 2. Einzelschritt eines Austauschschritts dar. Sind bei den Rechenoperationen

$$d_{ji}^{(m-1)} - d_{jm}^{(m-1)} \frac{d_{mi}^{(m-1)}}{d_{mm}^{(m-1)}} \text{ und } e_{ji}^{(m-1)} - e_{jm}^{(m-1)} \frac{e_{mi}^{(m-1)}}{d_{mm}^{(m-1)}}$$

die voneinander zu subtrahierenden Teile nahezu gleich – zum Beispiel gleich bis auf 13 Stellen –, so werden die Ergebnisse relativ ungenau, d. h. in diesem Fall maximal genau bis auf eine Stelle, wenn eine CD 3300 verwendet wird. Bei dem Pivotverfahren wird deshalb empfohlen, stets das betragsmässig grösste Element der Matrix als Pivotelement zu wählen[11], falls es noch nicht in einer Pivotzeile oder Pivotspalte stand. Da bei dem hier angegebenen Austauschverfahren die Pivotwahl mit der Hauptdiagonalen fest vorgegeben ist, führen Datenstrukturen mit grossen Absolutwerten in der Hauptdiagonalen zu relativ sicheren Ergebnissen.
Als weitere Kontrolle während des Programmtest kann die Ausgabe der Matrix D nach jedem Austauschschritt eingebaut werden, was durch die Anweisungsfolge

```
251 FOR I=1 TO N1
252   FOR J=1 TO N1
253     PRINT D(I,J),
254   NEXT J
255   PRINT
256 NEXT I

    ODER

255 MAT PRINT D

    ODER

255 MAT PRINT D,E
```

(eingefügt nach dem Statement 250) bewirkt wird. Eine Verbesserung des Verfahrens zur Erzielung einer höheren Rechengenauigkeit beim Vorliegen ungünstiger Datenstrukturen sei dem mathematisch interessierten Leser überlassen.
Verfügt der verwendete BASIC-Compiler über Matrizenanweisungen[12], so kann das obige Programm erheblich vereinfacht werden.

11 Dück, Numerische Methoden der Wirtschaftsmathematik I, S. 332–333.
12 Vgl. Abs. 331.

```
  5 REM MATRIZENINVERSION
 10 DIM D(10,10),E(10,10),P(10,10)
 20 READ N1
 30 DATA 4
 35 MAT LET D=ZER(N1,N1)
 40 MAT READ D
 45 PRINT #EINGABEMATRIX#
 50 MAT PRINT D
 60 MAT LET E=IDN(N1,N1)
 70 REM BEGINN DER INVERSION
 80 FOR M1=1 TO N1
 90   LET T=D(M1,M1)
100   IF T = 0 GOTO 310
105   REM BEGINN DES ERSTEN EINZELSCHRITTS
110   FOR N=1 TO N1
120     LET E(M1,N)=E(M1,N)/T
130     LET D(M1,N)=D(M1,N)/T
140   NEXT N
150   LET M=1
155   REM BEGINN DES ZWEITEN EINZELSCHRITTS
160   IF M1=M GOTO 240
170   LET T=D(M,M1)
180   FOR N=1 TO N1
210     LET E(M,N)=E(M,N)-E(M1,N)*T
220     LET D(M,N)=D(M,N)-D(M1,N)*T
230   NEXT N
240   LET M=M+1
250   IF M LE N1 GOTO 160
260 NEXT M1
265 REM ENDE DER INVERSION
270 PRINT
275 PRINT #INVERSE MATRIX#
280 MAT PRINT E
281 RESTORE N1**2
282 MAT READ D
283 MAT LET P=ZER(N1,N1)
284 MAT P=D*E
285 PRINT
286 PRINT #KONTROLLMATRIX#
287 MAT PRINT P
290 DATA 49.96, 12.23, 36.78, 29.43
292 DATA 22.46, 38.72, 40.15, 11.29
294 DATA 36.45, 19.20, 37.28, 6.43
296 DATA 17.84, 40.02, 27.86, 39.27
300 STOP
310 PRINT #EINGABEMATRIX IST SINGULAER#
320 END
```

Kennt der verwendete BASIC-Compiler die MAT-Funktion INV[13], so reduziert sich der Anweisungsumfang zur Inversion auf wenige Statements, wobei nun der Anwender allerdings keinen Einfluss mehr auf das intern verwendete Inversionsverfahren ausüben kann.

```
  5 REM MATRIZENINVERSION MIT DER MAT-FUNKTION INV
 10 DIM D(10,10),E(10,10),P(10,10)
 20 READ N1
 30 DATA 4
 35 MAT LET D=ZER(N1,N1)
 40 MAT READ D
 45 PRINT #EINGABEMATRIX#
 50 MAT PRINT D
 60 MAT LET E=ZER(N1,N1)
 70 MAT E=INV(D)
275 PRINT #INVERSE MATRIX#
280 MAT PRINT E
283 MAT LET P=ZER(N1,N1)
284 MAT P=D*E
286 PRINT #KONTROLLMATRIX#
287 MAT PRINT P
290 DATA 49.96, 12.23, 36.78, 29.43
292 DATA 22.46, 38.72, 40.15, 11.29
294 DATA 36.45, 19.20, 37.28, 6.43
296 DATA 17.84, 40.02, 27.86, 39.27
320 END
```

13 Vgl. Abs. 331.22

24 Programmalternativen

Wird die Matrizengleichung für ein lineares Gleichungssystem

$$Dx = p$$

wobei p der Spaltenvektor der rechten Seite und x der Lösungsvektor ist, mit der Kehrmatrix D^{-1} von links multipliziert

$$D^{-1}Dx = D^{-1}p$$

so ergibt sich daraus die Lösung des Gleichungssystems,

$$x = D^{-1}p$$

da $D^{-1}D$ eine Einheitsmatrix liefert.
Wird die Ausgabe der Kehrmatrix und die Kontrollrechnung im obigen BASIC-Programm – Statement 275 bis 287 – durch die Anweisungsfolge

```
275 MAT LET P=ZER(N,1)
280 MAT READ P
285 MAT LET P=E*P
287 PRINT ≠LOESUNG DES GLEICHUNGSSYSTEMS≠
288 MAT PRINT P
```

ersetzt, so erhält man ein Programm zur Lösung linearer Gleichungssysteme.
Ebenso kann eine einfache Modifikation des angegebenen Austauschverfahrens ohne Berechnung der Kehrmatrix zur Lösung linearer Gleichungssysteme führen.

```
  5 REM LÖSUNG LINEARER GLEICHUNGSSYSTEME
 10 DIM D(10,10)
 20 READ N1
 30 DATA 4
 40 LET J=N1+1
 50 MAT READ D(N1,J)
 60 PRINT ≠EINGABEMATRIX MIT RECHTER SEITE≠
 70 MAT PRINT D(N1,J)
 80 FOR M1=1 TO N1
 90   LET T=D(M1,M1)
100   IF T = 0 GOTO 300
110   FOR N=1 TO J
120     LET D(M1,N)=D(M1,N)/T
130   NEXT N
140   LET M=1
150   IF M = M1 GOTO 210
160   LET T=D(M,M1)
180   FOR N=1 TO J
190     LET D(M,N)=D(M,N)-D(M1,N)*T
200   NEXT N
210   LET M=M+1
220   IF M <= N1 GOTO 150
230 NEXT M1
235 PRINT
240 PRINT ≠LOESUNG DES GLEICHUNGSSYSTEMS≠
250 FOR M=1 TO N1
260   PRINT ≠X(≠;M;≠)≠,D(M,J)
270 NEXT M
280 STOP
290 DATA 49.96, 12.23, 36.78, 29.43, 40.43
292 DATA 22.46, 38.72, 40.15, 11.29, 15.50
294 DATA 36.45, 19.20, 37.28, 6.43, 42.40
296 DATA 17.64, 40.02, 27.86, 39.27, 25.57
300 PRINT ≠EINGABEMATRIX SINGULAER≠
310 END
```

```
EINGABEMATRIX MIT RECHTER SEITE
    MATRIX *D* ZEILE *   1*
       49.96          12.23          36.78          29.43          40.43
    MATRIX *D* ZEILE *   2*
       22.46          38.72          40.15          11.29          15.5
    MATRIX *D* ZEILE *   3*
       36.45          19.2           37.28           6.43          42.4
    MATRIX *D* ZEILE *   4*
       17.84          40.02          27.86          39.27          25.57

LOESUNG DES GLEICHUNGSSYSTEMS
X( 1 )                                 9.08816
X( 2 )                                 7.60192
X( 3 )                                -11.0839
X( 4 )                                -3.3612
```

Dem Leser sei empfohlen, hierzu die BASIC-Programmentwicklung analog zum Beispiel der Inversion selbständig durchzuführen.

3 BASIC-Struktur

31 Allgemeines

Im Rahmen dieser Arbeit wird generell zwischen Elementar- und Sonderanweisungen differenziert. Vgl. Abbildung 1. Sie lassen sich anhand ihrer spezifischen Kennworte unterscheiden.
Neben den bereits in Band 1 benutzten Elementarkennworten werden zur Darstellung von Sonderanweisungen zusätzlich Sonderkennworte benutzt. Es gilt mithin:

⟨Sprachanweisungs-kennwort⟩ 205 : = ⟨Elementarkennwort⟩ 74 | ⟨Sonderkennwort⟩ 205a

Die Sonderkennworte lassen sich entsprechend ihrer Anweisungszuordnung in Matrizen-, File-, erweiterte Programmsteuerungs-, Graphik- und Diagnosekennworte einteilen. Diese Gruppierung entspricht der Abschnittsgliederung in Band 2 und 3.

⟨Sonderkennwort⟩ 205a 205 : = ⟨Matrizenkennwort⟩ 205b | ⟨Filekennwort⟩ 205c |
⟨Erweitertes Programmsteuerungskennwort⟩ 205d |
⟨Graphikkennwort⟩ 205e | ⟨Diagnosekennwort⟩ 205f

Unter besonderer Berücksichtigung der FOSBIC Version 2 werden in Band 2 und 3 die folgenden Kennworte verwendet:

⟨Matrizenkennwort⟩ 205b 205a : = MAT | MAT INPUT | MAT LET | MAT LINPUT | MAT PRINT | MAT PRINT USING | MAT READ | §§§§

⟨Filekennwort⟩ 205c 205a : = AT END # | CLOSE # | IF EOF # | INPUT # | INQUIRE # | MAT

		INPUT #
		| MAT OUTPUT # | MAT WRITE #
		| OPEN # | OUTPUT # | SCRATCH #
		| WRITE # | §§§§
⟨Erweitertes Programmsteuerungskennwort⟩ 205d 205a	: =	CALL | CHAIN | LINK | SUB | SUBEXIT | SUBEND | §§§§
⟨Graphikkennwort⟩ 205e 205a	: =	CLEAR | END PICTURE | GINPUT | GPRINT | INQUIRE BOUNDS | INQUIRE BRUSH | INQUIRE CLIP | INQUIRE POSITION | INQUIRE VIEWPORT | INQUIRE WINDOW | PICTURE | PLOT | SET BOUNDS | SET BRUSH | SET CLIP | SET SET POSITION | SET VIEWPORT | SET WINDOW | §§§§
⟨Diagnosekennwort⟩ 205f 205a	: =	BREAK | DEBUG OFF | DEBUG ON DUMP | PRINT ALL | TRACE OFF TRACE ON | §§§§

Die Zeichenfolge §§§§ deutet jeweils an, dass die Anzahl der entsprechenden Kennworte den Entwicklungen neuer Standards und der Erweiterung des FOSBIC-Compilersystems angepasst werden kann.
Nach ANSI BASIC wird nicht zwischen Elementar- und Sonderkennworten unterschieden;[1] insgesamt werden über hundert Sprachanweisungskennworte ausgewiesen.[2]

1 Die Elementarkennworte wurden in ANSI Minimal BASIC vollumfänglich spezifiziert; vgl. Bd. 1, S. 124.

2 ANSI BASIC, p. 6: "The keywords in BASIC are: ACCESS, ALL, AND, AT, BASE, BEGIN, BOUNDS, BREAK, BRUSH, CALL, CASE, CHAIN, CLEAR, CLIP, CLOSE, COLLATE, CONNECT, DATA, DEBUG, DECIMAL, DELARE, DEF, DELAY, DIM, DISCONNECT, DISPLAY, ELSE, END, EOF, EVENT, EXIT, FILE, FOR, FROM, GINPUT, GO, GOSUB, GOTO, GRPRINT, IF, INPUT, INTEGER, INTERVAL, LENGTH, LET, LINPUT, LOOP, MAT, NAME, NATIVE, NEXT, OFF, ON, OPEN, OPTION, OR, ORGANIZATION, OUT, OUTIN, OUTPUT, PARACT, PAREND, PARSTOP, PICTURE, PLOT, POSITION, PRINT, PROCESS, PROMPT, RANDOMIZE, READ, REAL, RECORD, REM, RESTORE, RETURN, SCRATCH, SELECT, SEQUENTIAL, SET, SIGNAL, STANDARD, START, STATUS, STEP, STOP, SUB, SUBEND, SUBEXIT, THEN, TIME, TIMEOUT, TO, TRACE, TYPE, UNTIL, UR-

32 Elementaranweisungen

In Band 1 wurden lediglich BASIC-Elementaranweisungen behandelt. Sie stellen den Kern (Nukleus) der BASIC-Sprachanweisungen dar. Für die Elementaranweisungen gilt die Gleichung

⟨Elementar-anweisung⟩ := ⟨BASE-Statement⟩ | ⟨COMMENT-Statement⟩ |
206 204 97|98 84

⟨DATA-Statement⟩ | ⟨data-Statement⟩ |
110 114

⟨DEF-Statement⟩ | ⟨DIM-Statement⟩ |
100 86

⟨END-Statement⟩ | ⟨FOR/NEXT-Statement⟩ |
188|189|190 176|177|182

⟨GOSUB-Statement⟩ | ⟨GOTO-Statement⟩ |
191 168

⟨IF-Statement⟩ | ⟨INPUT-Statement⟩ |
172 117

⟨IMAGE-Statement⟩ | ⟨LET-Statement⟩ |
132 154|155|158

⟨let-Statement⟩ | ⟨LOOP/EXIT-Statement⟩ |
161 183|184|185

⟨null-Statement⟩ | ⟨PAGE-Statement⟩ |
82 152

⟨PRINT-Statement⟩ | ⟨PRINT TAB-Statement⟩ |
123 126

⟨PRINT USING-Statement⟩ | ⟨READ-Statement⟩ |
129 115

GENCY, USING, VARIABLE, VIEWPORT, WAIT, WHILE, WINDOW, and WITH". – Zu erwähnen ist, dass das Kennwort WHILE in keiner genormten ANSI BASIC-Anweisung Verwendung findet. – Nach ANSI BASIC 82 wird die Zahl der Kennworte wesentlich ausgeweitet; vgl. ANSI BASIC 82, p. 6.

⟨REM-Statement⟩ | ⟨RESTORE-Statement⟩ |
80|81 118|119|120|121

⟨RETURN-Statement⟩ |
197|198|198a

⟨STOP-Statement⟩
186|187

Diese Statements sind auch bei den meisten Minicomputern vollumfänglich verfügbar. Vgl. Tabelle 1.

33 Sonderanweisungen

Die Sonderanweisungen beginnen – falls von der Statementnummer abgesehen wird – generell mit einem Sonderkennwort; sie können im weiteren aber auch Elementarkennworte mitumfassen. Als typisches Beispiel sei die MAT READ-Anweisung erwähnt. Eine alphabetisch geordnete Zusammenstellung der in Band 2 und 3 behandelten Sonderanweisungen enthält Gleichung 384 (Band 3).[3]
Die Sonderanweisungen lassen sich zunächst funktionsbezogen weiter in Matrizen- und Fileanweisungen aufteilen. Sie werden – zusammen mit den nach ANSI BASIC 79 und 82 ebenfalls normierten – erweiterten Programmsteuerungsanweisungen in den Abs. 331, 332 und 333 behandelt. Die Graphik- und Diagnoseanweisungen gelangen in den Abs. 334 und 335 von Band 3 zur Darstellung.
Unberücksichtigt bleiben die in der Gleichung 206 durch §§§§ angedeuteten übrigen funktionsbezogenen Sonderanweisungen.

⟨Sonder-anweisung⟩ := ⟨Matrizen-anweisung⟩ | ⟨File-anweisung⟩ |
206 204 208 258

⟨Erweiterte Programm-steuerungsanweisungen⟩ |
295

Graphikanweisungen | ⟨Dignose-anweisung⟩ | ⟨§§§§
318 375

3 Vgl. Bd. 3, Kapitel 34.

Vermerkt sei, dass die in Band 1 vorgenommene Differenzierung zwischen nichtausführbaren und ausführbaren Anweisungen auch bezüglich der Sonderanweisungen Gültigkeit hat. Dies mit dem Ergebnis, dass sämtliche BASIC-Anweisungen nach der Gleichung

$$\underset{76}{\langle \text{Anweisung} \rangle} \underset{69}{:=} \left\langle \begin{matrix} \text{Nichtausführ-} \\ \text{bare Anweisung} \end{matrix} \right\rangle_{77} \Big| \left\langle \begin{matrix} \text{Ausführbare} \\ \text{Anweisung} \end{matrix} \right\rangle_{78}$$

klassifizierbar sind und dass die – in gewissem Masse immer willkürliche – Differenzierung zwischen Elementar- und Sonderanweisungen in der allgemeinen BASIC-Grundstruktur nicht zum Durchbruch kommt. Vgl. hierzu Band 3, Abs. 34.

Im folgenden wird die tabellarische Gegenüberstellung von BASIC-Compilern analog zu Band 1 fortgeführt. Dabei werden die Standardisierungsvorschläge gemäss ANSI BASIC im Sinne eines idealen BASIC-Prozessors mitberücksichtigt. Hinweise auf ANSI Minimal BASIC entfallen dagegen vollständig; auch der UWBIC-Compiler, der – ausser den beiden Diagnoseanweisungen DUMP und PRINT ALL – über keine Sonderanweisungen verfügt, wird in den folgenden Tabellen nicht mitgeführt. Neu in den Vergleich aufgenommen werden die BASIC-Prozessoren der Minicomputerhersteller:

Nixdorf Computer AG
Digital Equipment Corporation und
Wang Laboratories, Inc.

331 Matrizenanweisungen

Statt der bei Elementar-BASIC erforderlichen elementbezogenen Programmierung[4] lassen die MAT-Anweisungen die ganzheitliche Programmierung der Felderverarbeitung zu.[5]

Der Begriff Matrix umfasst im folgenden sowohl eindimensionale Tensoren (Vektoren) als auch zweidimensionale Tensoren (Matrizen).[6] Die

4 Vgl. hierzu Bd. 1, Abs. 222 und Bd. 1 Abbildung 8 und 10.

5 Garland, Dartmouth BASIC, p. 60: "It is... convenient to regard matrices as entities rather than as indexed collections of entities, and to manipulate the entire entity at once. BASIC provides a number of standard operations to facilitate such manipulations." – Ähnlich auch NCC Standard BASIC, p. 126; wörtliche Übernahme durch ANSI BASIC, p. 48.

6 Zum Matrizenbegriff vgl. Heinz Körth, Carl Otto, Walter Runge und Manfred Schoch. Lehrbuch der Mathematik für Wirtschaftswissenschaften. Opladen 1972, S. 143–150.

Begriffverwendung lehnt sich an das bei Matrizenanweisungen stets vorhandene Kennwort MAT an und stimmt insofern auch mit der ANSI BASIC-Norm überein. Wichtig erscheint jedoch der Hinweis, dass bei einzelnen BASIC-Programmiersystemen die Matrizenanweisungen teilweise nur auf zweidimensionale Felder anwendbar sind. Zum Beispiel führt die Anweisung

```
330 MAT A=ZER
```

bei Nixdorf Business Basic auf die Fehlermeldung

```
BASIC ERROR 33          (OPERAND IS NO MATRIX)
```

wenn der Vektor A mit

```
100 DIM A(25)
```

als eindimensionales Feld festgelegt wurde.
Sämtliche Matrizenanweisungen gehören zur Kategorie der ausführbaren Anweisungen, können aber zusätzlich noch Funktionen des DIM-Statements – einer nichtausführbaren Elementaranweisung[7] – übernehmen; vgl. Abb. 5

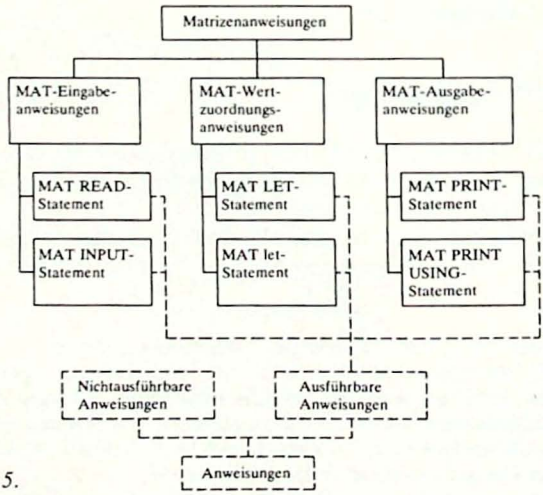


Abbildung 5.

7 Vgl. Bd. 1, Abs. 331.22.

Der vorstehenden Abbildung entspricht die Gleichung

$$\underset{208}{\langle \text{Matrizen-anweisung} \rangle} \underset{207}{:=} \underset{209}{\langle \text{MAT-Eingabe-anweisung} \rangle} \mid \underset{228}{\langle \text{MAT-Wertzu-ordnungs-anweisung} \rangle} \mid \underset{248}{\langle \text{MAT-Aus-gabe-anweisung} \rangle}$$

331.1 MAT-Eingabeanweisungen

Die MAT-Eingabeanweisungen können grundsätzlich als MAT READ- oder MAT INPUT-Statements realisiert werden, wobei die letzterwähnte Form für den Dialogbetrieb charakteristisch ist.

$$\underset{209}{\langle \text{MAT-Eingabe-anweisung} \rangle} \underset{208}{:=} \underset{210}{\langle \text{MAT READ-Statement} \rangle} \mid \underset{227}{\langle \text{MAT INPUT-Statement} \rangle}$$

331.11 MAT READ-Statements

MAT READ-Statements bewirken für die Elemente der in ihnen gelisteten Matrizen eine Wertzuweisung aus der mittels DATA-Statements organisierten Datei[8] und – bei deren Ausschöpfung – gegebenenfalls aus den mittels data-Statements bereitgestellten Daten.[8a]
Generell können vier verschiedene Formen des MAT READ-Statements unterschieden werden:

$$\underset{210}{\langle \text{MAT READ-Statement} \rangle} \underset{209}{:=} \underset{213}{\langle \text{DIM-gebundenes MAT READ-Statement} \rangle} \mid \underset{215}{\langle \text{DIM-begrenztes MAT READ-Statement} \rangle} \mid \underset{225}{\langle \text{DIM-umfassendes MAT READ-Statement} \rangle} \mid \underset{211}{\langle \text{Allgemeines MAT READ-Statement} \rangle}$$

8 NCC Standard BASIC, p. 149: "The math-read statement fills the array destination(s) in the mat-read list with Values obtained from the data block." – Vgl. auch Bd. 1, Abs. 322.11.

8a Vgl. Bd. 1, Abs. 332.12.

Diese MAT READ-Statements unterscheiden sich nur bezüglich der Feldspezifikation in dem MAT-Eingabelisten.
DIM-gebundene MAT READ-Statements sind dadurch charakterisiert, dass die MAT-Eingabeliste nur aus Feldnamen[8b] besteht, die via DIM-Statement(s) grössenordnungsmässig festgelegte und unverändert bleibende Matrizen bezeichnen.
Bei DIM-begrenzten MAT READ-Statements werden die Feldnamen durch DIM-Angaben ergänzt und damit für die einzelnen Matrizen eine Aktualisierung der via DIM-Statement festgelegten Zeilen-/Spaltenzahl erreicht.
DIM-umfassende MAT READ-Statements führen zur erstmaligen Dimensionierung der in ihnen aufgeführten Felder. Eine vorausgehende Dimensionierung mittels DIM-Statement entfällt.
Grundsätzlich müssen die in der MAT-Eingabeliste eines MAT READ-Statements spezifizierten Matrizen nicht unbedingt von gleichem Typ sein. Eine MAT-Eingabeliste kann sehr wohl gleichzeitig DIM-gebundene, DIM-begrenzte und – sofern zulässig – auch neu zu dimensionierende Matrizen betreffen. Allgemeine (hybride) MAT READ-Statements sind mithin zulässig; für sie gilt generell:

⟨Allgemeines MAT READ-Statement⟩ 211 : = 209

MAT READ ⟨MAT-Eingabeliste⟩ 212

⟨MAT-Eingabeliste⟩ 212 : = ⟨DIM-gebundene MAT-Eingabeliste⟩ 211 214 | ⟨DIM-begrenzte MAT-Eingabeliste⟩ 216 |
⟨DIM-umfassende MAT-Eingabeliste⟩ 226 | ⟨MAT-Eingabeliste⟩ 212 , ⟨MAT-Eingabeliste⟩ 212

8b Für den Feldnamen lassen die meisten BASIC-Compiler nur einen Buchstaben zu; nach ANSI BASIC (p. 45) soll jedoch auch die erweiterte Form mit Buchstabe und Ziffer eingeführt werden. – Vgl. Bd. 1, Abs. 323.12, 323.21.

Im folgenden werden die drei vorerwähnten (reinen) Grundformen des MAT READ-Statements einzeln behandelt; die für das allgemeine (hybride) MAT READ-Statement notwendigen Umsetzungen lassen sich leicht vornehmen und seien dem Leser überlassen.

Prozessor	Leseart	DIM-gebundenes MAT READ	DIM-begrenztes MAT READ	DIM-umfassendes MAT READ	MAT INPUT
ANSI BASIC	zeilen-weise	ja	ja	—	ja
Dartmouth BASIC-6	zeilen-weise	ja	ja	—	ja
UOM BASIC/ UBASIC	zeilen-weise	ja	—	ja	ja
IBM CALL BASIC	spalten-weise	ja	ja	—	—
FOSBIC	zeilen-weise	ja	ja	ja	ja
Wang Laboratories	zeilen-weise	ja	—	—	—
Nixdorf	zeilen-weise	ja	ja	ja	ja
Digital Equipment	zeilen-weise	ja	—	—	ja

Tabelle 5. Verfügbarkeit der MAT-Eingabeanweisungen

331.111 DIM-gebundenes MAT READ-Statement

Mit DIM-gebundenen MAT READ-Statements werden vorgängig dimensionierte und bezüglich der in den entsprechenden DIM-Statements angegebenen Zeilen- und Spaltenzahlen unverändert bleibende Matrizen eingelesen.

⟨DIM-gebundenes MAT READ-Statement⟩ : =
213 210

MAT READ ⟨DIM-gebundene MAT-Eingabeliste⟩ 214

$$\left\langle \begin{matrix}\text{DIM-gebundene}\\ \text{MAT-Eingabeliste}\end{matrix}\right\rangle_{214} := \langle\text{Feldname}\rangle_{40} \mid \langle\text{Feldname}\rangle_{40}\ \$ \mid \left\langle \begin{matrix}\text{DIM-gebundene}\\ \text{MAT-Eingabeliste}\end{matrix}\right\rangle_{214} , \left\langle \begin{matrix}\text{DIM-gebundene}\\ \text{MAT-Eingabeliste}\end{matrix}\right\rangle_{214}$$

$$\left\langle \begin{matrix}\text{DIM-gebundene}\\ \text{MAT-Eingabeliste}\end{matrix}\right\rangle_{214} \quad 212|213$$

Vermerkt sei, dass nach ANSI BASIC zur Definition des Feldnamens auch Grossbuchstaben und eine Ziffer zugelassen sind. Ebenso wird bei einzelnen BASIC-Compilern die Definitionszahl der Feldnamen durch die Anwendung des ausgeweiteten Alphabets erhöht;[9] diese Feldnamenform ist nach ANSI BASIC nicht vorgesehen und kann zu inkompatiblen BASIC-Programmen führen. Hier und im folgenden wird vereinfachend auf die Übernahme von

$$\langle\text{Feldname}\rangle_{40|41}$$

in die jeweiligen metalinguistischen Gleichungen verzichtet. Der Anwender kann davon ausgehen, dass die jeweils für die Elementaranweisungen gültige Form der Feldnamenbildung seines BASIC-Compilers auch bei den Sonderanweisungen bestehen bleibt.

Bei Verwendung von DIM-gebundenen MAT READ-Statements – einzelne BASIC-Compiler lassen nur diese Statementform zu – ist zum Einlesen der Matrix $X_{10\times 20}$ die Statementfolge

```
10 DIM X(10,20)
20 MAT READ X
```

erforderlich. Ein Vergleich mit dem entsprechenden Elementar-BASIC-Programm

```
10 DIM X(10,20)
20 FOR I=1 TO 10
30   FOR J=1 TO 20
40     READ X(I,J)
50   NEXT J
60 NEXT I
```

zeigt eine wesentliche Vereinfachung in der Programmstruktur[10]. Ebenso tritt im allgemeinen auch eine Verkürzung des vom Compiler erzeugten Objectcodes auf, was einer verringerten Zentralspeicherbeanspruchung bei gleichzeitiger Vereinfachung der Programmstruktur gleichkommt.

9 Vgl. Bd. 1, Abs. 323.12 und Abs. 323.22.

10 Zur vereinfachten Schleifendarstellung in den Programmablaufplänen vgl. Bd. 1, Abs. 222.

Der FOSBIC-Code benötigt für die Ausführung eines MAT-Statements der Form MAT READ A nur zwei Speicherplätze, anstelle von 32 (Vektor) bzw. 60 (Matrix) bei Benutzung der Elementaranweisungen[11]. Mit einem MAT READ-Statement können, wie das folgende Beispiel zeigt, auch mehrere Matrizen eingelesen werden.

```
10 DIM A(5), B(8,3), C(9,7)
20 MAT READ A, B, C
```

Beim Einlesen einer Matrix erfolgt die Zuweisung der Einzelwerte zeilenweise und in der Zeile von links nach rechts[12]. Darauf ist bei Anordnung der via DATA-Statements bereitzustellenden Daten sorgfältig zu achten.
Im folgenden Programm erfolgt die Dateneingabe – FOSBIC-konform – teilweise auch unter Verwendung eines data-Statements;

```
10 DIM A(3,4)
20 MAT READ A
30 PRINT ALL
40 DATA 1,2,3,4,5,6
99 END LIST DATA

LIST OF DATA AFTER END STATEMENT

7,8,9,10,11,12
```

Der mit dem PRINT ALL-Statement[13] erzeugte Ausdruck des Feldes A beweist die obige Speicherplatzbelegung.

A(1,1) = 1 A(1,2) = 2 A(1,3) = 3 A(1,4) = 4 } 1. Zeile	A(2,1) = 5 A(2,2) = 6 A(2,3) = 7 A(2,4) = 8 } 2. Zeile	A(3,1) = 9 A(3,2) = 10 A(3,3) = 11 A(3,4) = 12 } 3. Zeile

11 Mittels des DUMP-Statements kann dies vom Leser an einem Beispiel selbst überprüft werden, vgl. Bd. 3, Abs. 335.25.
12 NCC Standard BASIC, p. 149: "Each array is filled row by row, starting with the first (not the zeroth)."
13 Vgl. Bd. 3, Abs. 325.24.

```
LOCS=   324  LOCU=   345  ROWS=     4  COLUMNS=    5.  VARIABLE A
                    A(   6) OR A(  1,   1) =       1            DATA(  331)
                    A(   7) OR A(  1,   2) =       2            DATA(  332)
                    A(   8) OR A(  1,   3) =       3            DATA(  333)
                    A(   9) OR A(  1,   4) =       4            DATA(  334)
                    A(  11) OR A(  2,   1) =       5            DATA(  336)
                    A(  12) OR A(  2,   2) =       6            DATA(  337)
                    A(  13) OR A(  2,   3) =       7            DATA(  338)
                    A(  14) OR A(  2,   4) =       8            DATA(  339)
                    A(  16) OR A(  3,   1) =       9            DATA(  341)
                    A(  17) OR A(  3,   2) =      10            DATA(  342)
                    A(  18) OR A(  3,   3) =      11            DATA(  343)
                    A(  19) OR A(  3,   4) =      12            DATA(  344)
LOCS=   345  LOCU=   347  ROWS=     0  COLUMNS=   11.  VARIABLE B
LOCS=   347  LOCU=   349  ROWS=     0  COLUMNS=   11.  VARIABLE C
```

Die zeilenweise Dateneinlesung muss insbesondere auch bei Verarbeitung eines mehrere Matrizen umfassenden MAT READ-Statements sorgfältig beachtet werden, wie das Beispiel

```
10 DIM A(2), B(3,1), C(1,2), D(2,3)
20 MAT READ A, B, C, D
30 DATA 1,2,  1,2,3,  1,2,  1,2,3,4,5,6
```

mit $A = [1\ 2]$ $\quad B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ $\quad C = [1\ 2]$ $\quad$ und $\quad D = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

zeigt.

Nach einer – von den Computerherstellern weitgehend beachteten – Konvention werden bei Feld-Dimensionierung auf Basis Null[13a] Null-Spalte und Null-Zeile bei der Matrizenverarbeitung nicht verwendet.[14] Nichteinhaltung dieser Konvention führt zu inkompatiblen BASIC-Programmen.[15]

Vom FOSBIC-Compiler wird die vorerwähnte Konvention eingehalten.[16]

13a Zur Dimensionierung vgl. Bd. 1, Abs. 331.22.

14 NCC Standard BASIC, p. 149: "The mat-read statement does not affect row or column zero."

15 Paul W. Murrill and Cecil W. Smith. BASIC Programming. Scranton 1971, p. 123: "[In the discussion about MAT statements it is generally assumed] that zero subscripts are not allowed, which seems to be the trend. However, programs using MAT statements that run on a system that does not use the zero subscript will not run on a system that does. The converse is also true." – Nach ANSI BASIC 82 (p. 101, 103) ist die Kennwortkombination MAT LINE INPUT zu verwenden.

16 Vgl. hierzu auch die Ausführungen über die DIM-Statements, Bd. 1, Abs. 331.21.

Im allgemeinen können numerische und alphanumerische Variable in beliebiger Reihenfolge mit MAT READ-Statements verarbeitet werden. Einzelheiten zeigt Tabelle 5.
Der FOSBIC-Compiler verarbeitet den Programmausschnitt

```
10 DIM A$(5), B(5)
20 MAT READ A$, B
30 DATA /EINLESEN EINES VEKTORS/
40 DATA 1,2,3,4,5
```

korrekt. Da pro alphanumerische Variable jeweils fünf Zeichen eingelesen werden[17], sind zur vollständigen Übernahme der Textzeile „Einlesen eines Vektors" fünf Elemente erforderlich. Der Vektor A$(5) weist nach Verarbeitung des MAT READ-Statements den Inhalt

A(1) = EINLE A(2) = SEN E A(3) = INES
A(4) = VEKTO A(5) = RS

auf, wobei das letzte Element mit drei Leerstellen aufgefüllt ist.
Die neueren Entwicklungen der Textverarbeitung[16a] gestattet es auch, längere Zeichenketten in einem Element eines Vektors oder einer Matrix zu speichern. Ausschlaggebend für die maximal speicherbare Textlänge ist die Angabe der maximalen Stringlänge im DIM-Statement.[16b] In einem Textvektor kann durch ein DIM-gebundenes MAT READ-Statement jedem Element ein Textinhalt unterschiedlicher Länge eingegeben werden. Überschreitet die eingegebene alphanumerische Konstante mit ihrer Zeichenzahl die maximale Textlänge, so geht beim FOSBIC-Compiler der überzählige Textteil verloren. Compilerspezifisch kann aber auch eine Fehlermeldung erfolgen. Ist der eingegebene Text in seiner Zeichenzahl kürzer als die maximal aufnehmbare Zeichenzahl, so wird der restliche Speicherplatz jeweils mit Leerzeichen aufgefüllt. Das Textende wird für jedes Element durch ein Textendezeichen gekennzeichnet.
Wird in einem DIM-gebundenen MAT READ-Statement eine nicht explizit dimensionierte numerische oder alphanumerische Variable angegeben, so erfolgt implizite Dimensionierung.[16c] Fehler in DIM-gebundenen

16a Vgl. Bd. 1, Abs. 323.22.
16b Vgl. Bd. 1, Abs. 331.212.
16c Vgl. Bd. 1, S. 88 und Abs. 331.21. – Nach ANSI BASIC 82 (p. 2) ist – im Gegensatz zu ANSI Minimal BASIC – eine implizite Dimensionierung nicht mehr zulässig.
17 Vgl. Bd. 1, Abs. 323.21.

MAT READ-Statements werden vom FOSBIC-Compiler bei der Programmkompilierung wie folgt ausgewiesen:
– Kennwortfehler bei MAT:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

– Kennwortfehler bei READ:

```
        I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B    331
```

– Fehler im Feldnamen:

```
        I DO NOT UNDERSTAND THE FOLLOWING PART IN THE MAT STATEMENT ABOVE
B
SEE BASIC TEXTBOOK  B331.111
```

– Unpassendes DIM-Statement:

```
        VARIABLE "B" IS DIMENSIONED AS A VECTOR BUT YOU TREAT IT AS A MATRIX
        I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B 331.21
```

Bei der Programmausführung wird moniert:
– Ausgeschöpfte Datenliste:

```
PROGRAM STOPPED -- END OF DATA
                   ERROR CODE=  8
            ***** SEE BASIC TEXTBOOK =  A 332.13  *****
            ***** I WAS AT LINE NUMBER    0
```

331.112 DIM-begrenztes MAT READ-Statement

Die meisten BASIC-Compiler lassen – vgl. Tabelle 5 – in Verbindung mit dem MAT READ-Statement auch Feldredimensionierungen zu, wobei der für die einzelnen Felder im DIM-Statement maximal festgelegte Speicherplatz nicht überschritten werden darf (dynamic redimensioning).[18]
Die für den Benutzer besonders komfortablen MAT READ-Statements weisen folgende Struktur auf:

⟨DIM-begrenztes MAT READ-Statement⟩ : =
215 210

18 NCC Standard BASIC, p. 149; ANSI BASIC, p. 97–100.

MAT READ ⟨DIM-begrenzte MAT-Eingabeliste⟩ 216

Die Eintragungen nach den Kennworten MAT READ entsprechen im einfachsten Fall formal den Eintragungen in einem DIM-Statement.[19]

$$\underset{216}{\left\langle \begin{array}{l}\text{DIM-begrenzte}\\ \text{MAT-Eingabeliste}\end{array}\right\rangle} := \underset{217}{\left\langle \begin{array}{l}\text{Numerische}\\ \text{Eintragung}\end{array}\right\rangle} \mid \underset{218}{\left\langle \begin{array}{l}\text{Alphanumerische}\\ \text{Eintragung}\end{array}\right\rangle} \mid$$

$$\underset{212|215}{} \qquad \underset{216}{\left\langle \begin{array}{l}\text{DIM-begrenzte}\\ \text{MAT-Eingabeliste}\end{array}\right\rangle}, \underset{216}{\left\langle \begin{array}{l}\text{DIM-begrenzte}\\ \text{MAT-Eingabeliste}\end{array}\right\rangle}$$

Für die numerischen und alphanumerischen Eintragungen treffen die generellen Regeln zum Aufbau von indizierten numerischen und alphanumerischen Variablen zu.[19a] Im speziellen gelten die folgenden Strukturgleichungen:

$$\underset{217}{\left\langle \begin{array}{l}\text{Numerische}\\ \text{Eintragung}\end{array}\right\rangle} := \underset{40}{\langle \text{Feldname} \rangle} \; (\underset{219}{\langle \text{Index-Trimmer} \rangle}) \qquad \underset{216}{}$$

$$\underset{218}{\left\langle \begin{array}{l}\text{Alphanumerische}\\ \text{Eintragung}\end{array}\right\rangle} := \underset{40}{\langle \text{Feldname} \rangle} \; \$ \; (\underset{219}{\langle \text{Index-Trimmer} \rangle}) \mid \qquad \underset{216}{}$$

$$\underset{40}{\langle \text{Feldname} \rangle} \; \$ \; (\underset{219}{\left\langle \begin{array}{l}\text{Index-}\\ \text{Trimmer}\end{array}\right\rangle}) \quad \underset{223}{\left\langle \begin{array}{l}\text{Stringlängen-}\\ \text{Trimmer}\end{array}\right\rangle} \mid$$

$$\underset{40}{\langle \text{Feldname} \rangle} \; \$ \; \underset{223}{\left\langle \begin{array}{l}\text{Stringlängen}\\ \text{Trimmer}\end{array}\right\rangle} \quad (\underset{219}{\left\langle \begin{array}{l}\text{Index-}\\ \text{Trimmer}\end{array}\right\rangle})$$

19 Vgl. Bd. 1, Abs. 331.21.
19a Vgl. Bd. 1, Abs. 323.12, 323.22.

Die Trimmer[20] dienen zur dynamischen Redimensionierung, wobei der grundsätzliche Aufbau der Index- und Stringlängen-Trimmer durch die Struktur der DIM-Statements vorgegeben ist. Vgl. Bd. 1, Gleichung 88 (Dimensionierung) und 93 (maximale Stringlänge). Allgemein gilt:

$$\langle\text{Index-Trimmer}\rangle_{219} := \left\langle\begin{matrix}\text{DIM-begrenzte}\\ \text{Zeilenzahl}\end{matrix}\right\rangle_{220} \;\Big|\; \left\langle\begin{matrix}\text{DIM-begrenzte}\\ \text{Spaltenzahl}\end{matrix}\right\rangle_{221}$$

$$\left\langle\begin{matrix}\text{DIM-begrenzte}\\ \text{Zeilenzahl}\end{matrix}\right\rangle_{220} , \left\langle\begin{matrix}\text{DIM-begrenzte}\\ \text{Spaltenzahl}\end{matrix}\right\rangle_{221}$$

Nach ANSI BASIC sind numerische und alphanumerische Eintragungen identisch genormt; Stringlängen-Trimmer sind nicht vorgesehen.[20a] Ausserdem bewirkt ein eingliedriger Index-Trimmer stets die Redimensionierung eines Spaltenvektors[20b] BASIC-Programme, bei denen diese Konvention nicht eingehalten wird, sind nur in beschränktem Masse transportfähig.

Die DIM-begrenzten Zeilen- und Spaltenzahlen errechnen sich allgemein aus einem arithmetischen Ausdruck, der nach ANSI BASIC zur nächsten Ganzzahl gerundet werden soll.[20c] Die redimensionierte Elementzahl darf die ursprünglich festgelegte maximale Elementzahl nicht überschreiten; es gilt daher:[20d]

$$\left\langle\begin{matrix}\text{DIM-begrenzte}\\ \text{Zeilenzahl}\end{matrix}\right\rangle_{220} := \left\langle\begin{matrix}\text{Arithmetischer}\\ \text{Ausdruck}\end{matrix}\right\rangle_{49} \leq \left.\frac{\text{DIM-Wert}_{222}}{\text{DIM-begrenzte Spaltenzahl}_{221}}\right\rangle_{219}$$

20 Die Bezeichnung „Trimmer" wurde von NCC Standard BASIC (p. 149) übernommen: "The trimmer allows dynamic redimensioning." Nach ANSI BASIC (p. 97) wird der entsprechende Teil dem DIM-begrenzten MAT-Eingabeliste „redim" genannt. – Nach ANSI BASI 82 (p. 49, 51) ist auch ein spezielles REDIM-Statement vorgesehen.

20a ANSI BASIC, p. 37.

20b Vgl. Bd. 1, S. 136; ANSI BASIC, p. 49: "A singly-subscripted numeric-array shall be treated as a column vector when used in a numeric-array-expression."

20c ANSI BASIC, p. 19: "An index is a numeric-expression whose value shall be rounded to the nearest integer."

20d ANSI BASIC, p. 49, 53.

$$\left\langle \begin{matrix}\text{DIM-begrenzte}\\ \text{Spaltenzahl}\end{matrix} \right\rangle_{221\ \ 219} := \left\langle \begin{matrix}\text{Arithmetischer}\\ \text{Ausdruck}\end{matrix} \right\rangle_{49} \leq \left. \frac{\text{DIM-Wert}_{222}}{\text{DIM-begrenzte Zeilenzahl}_{220}} \right\rangle$$

$$\langle \text{DIM-Wert} \rangle_{222\ \ 220|221} := \langle \text{Zeilenzahl} \rangle_{89} * \langle \text{Spaltenzahl} \rangle_{90}$$

Im folgenden Beispiel weist der Index-Trimmer im Vergleich zu der im DIM-Statement angegebenen Zeilen- und Spaltenzahl reduzierte Werte auf; statt der ursprünglich dimensionierten $A_{6\times6}$-Matrix wird die $A_{2\times3}$-Matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

eingelesen.

```
10 DIM A(6,6)
20 MAT READ A(2,3)
30 DATA 1,2,3,4,5,6,
```

Ohne Überschreitung des DIM-Werts wird im folgenden Programm mit vergrösserter Spaltenzahl gearbeitet und die $A_{2\times8}$-Matrix

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{bmatrix}$$

```
10 DIM A(6,6)
20 MAT READ A(2,8)
30 DATA 1,2,3,4,5,6,7,8
40 DATA 9,10,11,12,13,14,15,16
```

eingelesen.

Bei einzelnen Compilern kann für die DIM-begrenzte Zeilen- und Spaltenzahl nur eine einfache numerische Variable angegeben werden, so dass eventuell einem DIM-begrenzten MAT READ-Statement zunächst ein oder zwei LET-Statements vorausgestellt werden müssen, in denen einfachen numerischen Variablen die Ergebnisse von komplexen arithmetischen Ausdrücken zugewiesen werden.
Der FOSBIC-Compiler lässt die Festlegung der DIM-begrenzten Zeilenzahl und Spaltenzahl mittels arithmetischer Ausdrücke zu. Dabei dürfen innerhalb eines arithmetischen Ausdrucks auch Elemente der im MAT

READ-Statement redimensionierten Matrix vorkommen, da bei Abarbeitung des MAT READ-Statements zuerst die Zeilen- und Spaltenzahl berechnet wird.[20e] Bedingung für eine sinnvolle Redimensionierung ist, dass das Ergebnis des arithmetischen Ausdrucks positiv und grösser/gleich Eins ist. Ein eventueller Dezimalteil wird abgeschnitten.
Ein FOSBIC-konformer Programmausschnitt könnte beispielsweise wie folgt lauten:

```
10 DIM A(5,5), B(10,10)
20 READ M, N
30 MAT READ A(M,N), B(M,N)
40 DATA 2,3
50 DATA 1,2,3,4,5,6,10,20,30,40
```

Es erfolgt das Einlesen der Matrizen.

$$A_{2x3} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad B_{2x2} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

In gleicher Weise können auch Felder redimensioniert werden, die in der Folge mit alphanumerischen Werten zu belegen sind.
Der Programmausschnitt

```
10 DIM Z$(10)
20 READ M
30 MAT READ Z$(M)
40 DATA 3
50 DATA #FOSBIC-COMPILER#
```

führt zum Einlesen von Z$(1) = FOSBI, Z$(2) = C-COM und Z$(3) = PILER.
Vermerkt sei auch, dass ein ursprünglich als Vektor dimensioniertes Feld nachträglich – unter Einhaltung des DIM-Werts – zur Aufnahme beliebiger Matrizen redimensioniert werden kann. Vgl. hierzu das folgende Beispiel.

```
10 DIM A(25), B(100)
20 READ M,N
30 MAT READ A(M,N), B(M,N)
40 DATA 2,3
50 DATA 1,2,3,4,5,6,10,20,30,40
```

20e Vgl. auch ANSI BASIC, p. 98: "If a redim is present then dynamic redimensioning shall take place before values are assigned to the redimensioned array. Redims in the redim-array-list shall be evaluated after values have been assigned to the arrays preceding them (i. e., to the left of them) in the redim-array-list."

Ebenso ist die Redimensionierung einer Matrix als Vektor denkbar. Der FOSBIC-Compiler lässt den frei redimensionierbaren Übergang zwischen Vektoren und Matrizen nur zu, wenn die Vektoren als einzeilige oder einspaltige Matrizen im DIM-Statement definiert sind, vgl. hierzu folgendes Beispiel:

```
 10 DIM A(1,20), B(10,1), C(5,5)
 20 READ M,N
 30 DATA 3,2
 40 MAT LET A=ZER(M,N)
 50 MAT LET B=CON(N,1)
 60 MAT READ A
 70 DATA 1,2,3,4,5,6
 80 MAT LET C=ZER(M,1)
 90 MAT LET C=A*B
100 MAT LET A=ZER(1,M+2)
110 MAT READ A
   ...
   ...
```

Zu beachten ist, dass durch Redimensionierung einer Matrix die ursprünglich via DIM-Statement festgelegte Feldgrösse nur temporär verändert wird; die spätere Verarbeitung DIM-begrenzter MAT READ-Statements erfolgt nach den Spezifikationen des DIM-Statements. So wird im folgenden Beispiel

```
 10 DIM A(4,4)
 20 MAT READ A(2,3)
 30 PRINT ALL
 40 MAT READ A
 50 PRINT ALL
 60 DATA 1,2,3,4,5,6
 70 DATA 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,
 80 END
```

zunächst die Matrix $A_{2\times 3} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ eingelesen und verarbeitet; später (Statement 40) erfolgt die nochmalige Einlesung der Matrix A, nunmehr – gemäss DIM-Statement – als

$$A_{4x4} = \begin{bmatrix} 16 & 15 & 14 & 13 \\ 12 & 11 & 10 & 9 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

was durch den mittels PRINT ALL[21] erzeugten Ausdruck zu obigem Programmbeispiel bestätigt wird.

21 Zum PRINT ALL-Statement vgl. Bd. 3, Abs. 335.24.

```
LOCS=   324  LOCU=   350  ROWS=     5  COLUMNS=    5.  VARIABLE A
                    A(   6) OR A(   1,   1) =       1            DATA(  331)
                    A(   7) OR A(   1,   2) =       2            DATA(  332)
                    A(   8) OR A(   1,   3) =       3            DATA(  333)
                    A(  11) OR A(   2,   1) =       4            DATA(  336)
                    A(  12) OR A(   2,   2) =       5            DATA(  337)
                    A(  13) OR A(   2,   3) =       6            DATA(  338)
LOCS=   350  LOCU=   352  ROWS=     0  COLUMNS=   11.  VARIABLE B

LOCS=   324  LOCU=   350  ROWS=     5  COLUMNS=    5.  VARIABLE A
                    A(   6) OR A(   1,   1) =      16            DATA(  331)
                    A(   7) OR A(   1,   2) =      15            DATA(  332)
                    A(   8) OR A(   1,   3) =      14            DATA(  333)
                    A(   9) OR A(   1,   4) =      13            DATA(  334)
                    A(  11) OR A(   2,   1) =      12            DATA(  336)
                    A(  12) OR A(   2,   2) =      11            DATA(  337)
                    A(  13) OR A(   2,   3) =      10            DATA(  338)
                    A(  14) OR A(   2,   4) =       9            DATA(  339)
                    A(  16) OR A(   3,   1) =       8            DATA(  341)
                    A(  17) OR A(   3,   2) =       7            DATA(  342)
                    A(  18) OR A(   3,   3) =       6            DATA(  343)
                    A(  19) OR A(   3,   4) =       5            DATA(  344)
                    A(  21) OR A(   4,   1) =       4            DATA(  346)
                    A(  22) OR A(   4,   2) =       3            DATA(  347)
                    A(  23) OR A(   4,   3) =       2            DATA(  348)
                    A(  24) OR A(   4,   4) =       1            DATA(  349)
LOCS=   350  LOCU=   352  ROWS=     0  COLUMNS=   11.  VARIABLE B
```

Eine dynamische Redimensionierung der Feldgrenzen, die nicht nur für das jeweilige MAT-Statement zutrifft, ist nur mittels der MAT-Funktion durch MAT-Wertzuordnungsanweisungen möglich.[22]
Vom FOSBIC-Compiler wird bei DIM-begrenzten MAT READ-Statements in der Compilierungsphase nachgewiesen:

- Fehler im arithmetischen Ausdruck des Index-Trimmers: Übliche Fehlerdiagnose für arithmetische Ausdrücke.[23]
- Fehlende Klammer vor oder hinter dem Index-Trimmer:

```
                    MISSING PARENTHESIS IN MAT STATEMENT
                    THE PART I DO NOT LIKE IS
          SEE BASIC TEXTBOOK  B331.112
```

Bei der Programmarbeitung werden folgende Fehler moniert:

- Unrichtige Redimensionierung, Zeilenzahl:

```
PROGRAM STOPPED -- ROW INDEX OF THE MATRIX *A* IS NEGATIVE OR ZERO   -12345.00
                   ERROR CODE= 33
             ***** SEE BASIC TEXTBOOK =  B331.312  *****
             ***** I WAS AT LINE NUMBER    0
```

- Unrichtige Redimensionierung, Spaltenzahl:

```
PROGRAM STOPPED -- COLUMN INDEX OF THE MATRIX *A* IS NEGATIVE OR ZERO   -12345.00
                   ERROR CODE= 32
             ***** SEE BASIC TEXTBOOK =  B331.312  *****
             ***** I WAS AT LINE NUMBER    0
```

22 Vgl. Abs. 331.2
23 Vgl. Bd. 1, Abs. 324.1.

– Unrichtige Redimensionierung, zu hoher DIM-Wert:

```
PROGRAM STOPPED -- ILLEGAL REDIMENSION OF THE MATRIX "A" ROWS=
                   24 COLUMNS=   25  MAXIMAL STORAGE ELEMENTS 12345.
                   ERROR CODE= 34
              ***** SEE BASIC TEXTBOOK =  B331.312  *****
              ***** I WAS AT LINE NUMBER    8
```

Obwohl nach ANSI BASIC ein Stringlängen-Trimmer nicht vorgesehen ist, scheint es doch sinnvoll, eine erweiterte Textverarbeitung in den MAT READ-Statements zu antizipieren.[23a] Im folgenden wird für den Stringlängen-Trimmer eine Struktur analog zu Gleichung 93 vorgesehen, wobei die maximale Stringlänge als DIM-Stringlänge bezeichnet wird.[23b] Im übrigen gilt:

$$\underset{223}{\langle \text{Stringlängen-Trimmer} \rangle} \underset{218}{:=} (\underset{224}{\langle \text{Untere Längen-begrenzung} \rangle} : \underset{224a}{\langle \text{Obere Längen-begrenzung} \rangle})$$

$$\underset{224}{\langle \text{Untere Längen-begrenzung} \rangle} \underset{223}{:=} \underset{49}{\langle \text{Arithmetischer Ausdruck} \rangle} < \underset{224b}{\langle \text{DIM-String-länge} \rangle}$$

$$\underset{224a}{\langle \text{Obere Längen-begrenzung} \rangle} \underset{223}{:=} \underset{49}{\langle \text{Arithmetischer Ausdruck} \rangle} < \underset{224b}{\langle \text{DIM-String-länge} \rangle}$$

$$\underset{224b}{\langle \text{DIM-Stringlänge} \rangle} \underset{224|224a}{:=} \underset{93}{\langle \text{Maximale Stringlänge} \rangle}$$

Analog zur DIM-begrenzten Zeilen- und Spaltenzahl dürfen die arithmetischen Ausdrucke des Stringlängen-Trimmers nicht grösser als die ursprünglich maximal zulässige Stringlänge je Element werden, wobei ein gebrochen rationaler Zahlenwert zur nächsten Ganzzahl gerundet wird. Die FOSBIC Version 2 realisiert bereits diese Variante des MAT READ-Statements.
Als Beispiel für eine alphanumerische Eintragung in einem DIM-begrenzten MAT READ-Statement seien die folgenden Anweisungen angeführt, wobei die erste Anweisung der ANSI BASIC-Norm ohne Stringlängen-Trimmer entspricht und die zweite zwei mögliche Formen mit Stringlängen-Trimmer enthält.

23a Realisiert gemäss ANSI BASIC 82, p. 56–57.
23b Vgl. Bd. 1, S. 137.

```
100 MAT READ A$(3,4)
200 MAT READ B$(N,M)(1:8)
300 MAT READ C$(3,L*2)(X:Y)
```

Der Einsatz des Stringlängen-Trimmers kann erheblich zur Verkürzung von BASIC-Programmen beitragen. Zum Beispiel wird mit dem in ANSI BASIC-Norm gehaltenen Programm

```
10 DIM A$(20)30,B$(20)30
20 MAT A$="EINGE"
30 MAT READ B$(4)
40 DATA "ARBEITET", "FUEGT", "LESEN", "SETZT"
50 FOR I=1 TO 4
60    A$(I)(6:30)=B$(I)
70 NEXT I
80 MAT PRINT A$
90 END
```

die Druckausgabe

```
EINGEARBEITET  EINGEFUEGT     EINGELESEN     EINGESETZT
```

erzeugt. Wesentlich kürzer gestaltet sich das Programm mit einem Stringlängen-Trimmer.

```
10 DIM A$(20)30
20 MAT A$="EINGE"
30 MAT READ A$(4)(6:30)
40 DATA "ARBEITET", "FUEGT", "LESEN", "SETZT"
50 MAT PRINT A$
60 END
```

331.113 DIM-umfassendes MAT READ-Statement

Einzelne Compiler lassen auch die erstmalige Dimensionierung von Matrizen im MAT READ-Statement zu; vgl. Tabelle 5. In allgemeinster Form gilt für dieses MAT READ-Statement:

⟨DIM-umfassendes MAT READ-Statement⟩ : =
225 210

MAT READ ⟨DIM-umfassende MAT-Eingabeliste⟩ 226

$$\left\langle \begin{matrix}\text{DIM-umfassende}\\ \text{MAT-Eingabeliste}\end{matrix} \right\rangle := \left\langle \begin{matrix}\text{Numerische}\\ \text{Deklaration}\end{matrix} \right\rangle \mid \left\langle \begin{matrix}\text{Alphanumerische}\\ \text{Deklaration}\end{matrix} \right\rangle$$

226 212|225 87 95

Aus der vorstehenden metalinguistischen Gleichung ergibt sich, dass eine DIM-umfassende Liste als implizites DIM-Statement interpretiert werden kann.
Die erstmalige Dimensionierung einer Matrix im Rahmen eines MAT READ-Statements kann – wie beim DIM-Statement – nur in Form vorzeichenloser Ganzzahlen erfolgen.[24] Nach ANSI BASIC ist das DIM-umfassende MAT READ-Statement noch nicht vorgesehen.[24a]
Im folgenden Beispiel werden die Matrizen A und B erstmals im MAT READ-Statement dimensioniert.

```
10 DIM C(2,2)
20 MAT READ A(2,3), B(3,2)
30 DATA 2,2,2,2,2,2
40 DATA 3,3,3,3,3,3
```

Es werden die Matrizen

$$A = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \text{ und } B = \begin{bmatrix} 3 & 3 \\ 3 & 3 \\ 3 & 3 \end{bmatrix}$$

eingelesen.
Der FOSBIC-Compiler lässt auch DIM-umfassende MAT READ-Statements zu. Falls die erstmalige Dimensionierung – fälschlicherweise – durch einen arithmetischen Ausdruck erfolgt, wird dies vom FOSBIC-Compiler mit dem Hinweis

```
DIMENSION NOT ALLOWED WITH EXPRESSION
I DO NOT UNDERSTAND THE FOLLOWING PART IN THE MAT STATEMENT ABOVE

SEE BASIC TEXTBOOK  R331.211
```

ausgewiesen. Ansonsten können bei der Verarbeitung DIM-umfassender MAT READ-Statements dieselben Fehlermeldungen wie beim DIM-begrenzten MAT READ-Statement auftreten.

24 Vgl. Bd. 1, Abs. 332.211, 332.212. – Bei Nixdorf können sich die Dimensionsgrenzen auch aus arithmetischen Ausdrücken ergeben; vgl. Nixdorf, System-Software Business-Basic, S. 10.21.

24a Nach ANSI BASIC 82 wird es ausgeschlossen; vgl. ANSI BASIC 82, p. 2: "All arrays in a standard conforming program must be dimensioned before use."

331.12 MAT INPUT-Statements

Auch die MAT INPUT-Statements dienen zur ganzheitlichen Eingabe von Matrizen;[25] ihre Strukturen sind denen des elementaren INPUT-Statements vergleichbar[26], wobei hier die Variablenliste durch die MAT Inputliste ersetzt wird.

⟨MAT INPUT-Statement⟩ : =
227 209

MAT INPUT ⟨MAT Inputliste⟩ | 227a
MAT INPUT ⟨PROMPT-Identifikation⟩ ⟨MAT Inputliste⟩   117b 227a

Die MAT Inputliste unterscheidet sich von der MAT-Eingabeliste des allgemeinen MAT READ-Statements nur durch die zusätzliche Möglichkeit, einen Vektor mit variabler Elementzahl einzugeben.

⟨MAT Inputliste⟩ : = ⟨Variabler Vektor⟩ | ⟨MAT-Eingabeliste⟩
227a 227a 227b 212

⟨MAT-Eingabeliste⟩ , ⟨MAT-Inputliste⟩
212 227a

Der variable Vektor ist nach folgender Struktur anzugeben:

⟨Variabler Vektor⟩ : = ⟨Feldname⟩ (?) | ⟨Feldname⟩ $ (?)
227b 227a 40 40

Dabei handelt es sich nach ANSI BASIC[27] immer um einen Zeilenvektor.

26 Vgl. Bd. 1, Abs. 332.14.
27 ANSI BASIC, p. 97.

Die MAT INPUT-Statements unterscheiden sich in ihrer Funktionsweise dadurch von den entsprechend aufgebauten MAT READ-Statements, dass sie speziell auf den Dialogbetrieb ausgerichtet sind und die Eingabe der Matrizenelemente direkt via Terminal verlangen.[28] Dem entsprechend entfällt bei ihnen – wie beim einfachen INPUT-Statement[29] – der Aufbau einer Datei über DATA-Statements und deren spätere Benutzungsmöglichkeit unter Einschaltung von RESTORE-Statement.[30] Über ein MAT INPUT-Statement kann auch die Eingabe mehrerer Matrizen angefordert werden.

```
MAT INPUT A, B$
```

Aus dem vorstehenden Beispiel ergibt sich, dass MAT INPUT-Statements auch zur Belegung alphanumerischer Felder verwendet werden können[31], wobei die via Terminal einzugebenden Strings bei einzelnen BASIC-Compiler zudem nicht unbedingt in Stringklammern gesetzt sein müssen.[32]
Generell werden die im Anschluss an ein MAT INPUT-Statement einzugebenden Matrizen – durch Ausgabe von Fragezeichen am Terminal – zeilenweise angefordert. Die entsprechenden Einzelwerte sind vom Programmbenutzer jeweils vektorweise von links nach rechts – je durch ein Komma getrennt – einzugeben, wobei dieser Prozess so lange abläuft, bis die Matrix vollständig aufgefüllt ist.[32a]
Das MAT INPUT-Statement wird vorzugsweise in Kombination mit einem vorgeschalteten (zur Unterbindung des Zeilenvorschubs mit; abge-

28 NCC Standard BASIC, p. 139: "The mat-input statement fills the array destinations appearing in it with values obtained at run-time from the terminal (as with the ordinary input statement)." – Vgl. auch ANSI BASIC, p. 98: "Execution of an array-input-statement shall cause execution of the program to be suspended until a valid input-reply... has been supplied. After validation of an input-reply, an array-input-statement shall cause arrays in the redim-array-list to be assigned, in order, values from the input-reply. Values shall be assigned to all elements in each array row by row."
29 Vgl. Bd. 1, Abs. 332.14.
30 Vgl. Bd. 1, Abs. 332.15.
31 Waite and Mather, BASIC, p. 60: "The MAT INPUT statement may also be used to enter strings into a list."
32 Vgl. Bd. 1, Abs. 332.2. – Zu Dartmouth BASIC-6 im besonderen: Waite and Mather, BASIC, p. 60–61.
32a Vgl. ANSI BASIC, p. 98: "Data in response to a request for array input need not be supplied in a single input-reply. If an input-reply contains a final comma, then a further input-reply shall be requested to obtain more data."

schlossen) PRINT-Statement verwendet, um den Programmbenutzer via Terminal über die Art der Eingabedaten genau zu informieren.
Die vorliegende Norm der MAT Inputliste hat bei der Dateneingabe in Matrizen den Nachteil, dass der Anwender auf dem Bildschirm nicht feststellen kann, in welches Matrizenelement gerade die Eingabe erfolgt, wenn er die Anzahl der eingegebenen Elemente Datenelemente nicht mitzählt. Ein weiterer Nachteil ergibt sich im Zusammenhang mit dem MAT INPUT FL-Statement.[32b]
Bei Verarbeitung des folgenden Programmausschnitts

```
10 DIM P(3)
20 PRINT #WELCHE DREI WERTE FUER VEKTOR P#
30 MAT INPUT P
```

erscheint am Terminal die Eingabeanweisung

```
WELCHE DREI WERTE FUER VEKTOR P ↵
```

Die Steuerung der Dateneingabe kann in gewissem Umfange durch den Programmbenutzer vom Terminal aus erfolgen. So wird bei Dartmouth BASIC-6 die erwünschte Fortsetzung der Eingabeliste auf einer neuen Zeile durch das &-Zeichen angezeigt und vom Compiler durch Ausgabe eines neuen Fragezeichens auf der Folgezeile quittiert.[33] Vgl. hierzu das folgende Beispiel.

```
10 DIM V(5)
20 MAT INPUT V
```

Die bisher angeführten Beispiele betrafen ausschliesslich DIM-gebundene MAT INPUT-Statements. Besonders benutzerfreundlich sind die DIM-begrenzten MAT INPUT-Statements. Sie können analog zu den entsprechenden MAT READ-Statements strukturiert werden.

```
10 DIM M(2,12)
20 MAT INPUT M(3,8)
```

Im vorstehenden Beispiel wurde über das DIM-Statement Speicherplatz für eine $M_{2\times12}$-Matrix reserviert; via MAT INPUT-Statement soll aber eine $M_{3\times8}$-Matrix eingelesen werden.[34]

32b Vgl. Abs. 332.43. – Dort wird ein Vorschlag zur Erweiterung der MAT Inputliste diskutiert, der diese Nachteile vermeidet.
33 Vgl. Waite and Mather, BASIC, p. 59–61.
34 Vgl. auch: Waite and Mather, BASIC, p. 118–120.

Die in Gleichung 227 mitenthaltene PROMPT-Identifikation entspricht vollständig derjenigen des elementaren INPUT-Statements.[34a] Die Angabe in der PROMPT-Identifikation erscheint auch beim MAT INPUT-Statement nur am Anfang der Eingabe und wird nicht nach der Eingabe jedes Datums wiederholt.
Eine programmtechnisch äusserst sinnvolle Modifikation der MAT-Eingabeliste stellt die Angabe des variablen Vektors dar. Allgemein fällt diese Angabe noch unter die DIM-begrenzte Eingabeliste, wobei hier jedoch die Anzahl der Elemente nicht programmintern festgelegt wird, sondern programmextern durch den Anwender geschieht.
Bei Verwendung des variablen Vektors wird die Eingabe von Daten solange erwartet, bis entweder die maximal dimensionierte Elementzahl erreicht ist oder das Eingabeende durch den Benutzer angezeigt wird. Die dynamische Redimensionierung erfolgt dann nach der Anzahl der eingegebenen Daten. Nach ANSI BASIC wird das Ende einer Eingabefolge mit variablem Vektor dadurch angezeigt, dass nach dem letzten Eingabewert *kein* Komma folgt.
Zur Feststellung der über einen variablen Vektor eingegebenen Daten dient nach ANSI BASIC die – schon in NCC Standard BASIC diskutierte[35] – Funktion NUM.[35a] Vgl. hierzu das folgende Beispiel, bei dem die Zahl der mit NUM erfassten Eingabewerte beim FOR/NEXT-Statement als Laufbereichsgrenze verwendet wird.[36]

34a Vgl. Bd. 1, S. 168; ANSI BASIC, p. 98: "If a prompt-specifier is present in the array-input-statement, then the implementation-defined array-input-prompt shall not be output; instead, the value of the string-expression in the prompt-specifier shall be output to inform the user of the program of the need to supply data. In the batch mode, the imput-replay shall be requested from the external source by an implementation-defined means."

35 NCC Standard BASIC, p. 139: "If the only destination [of a mat-input statement] is a vector without a trimmer... input of any number of values, from zero to the current size of the vector destination is allowed. If the values do not fit on to one line, the line may be ended with a comma; another line of input will then be called for. There is no limit to the number of lines which may end in a comma. After such a mat-input statement, the parameterless function NUM gives the number of values input, and the vector destination is redimensioned to this size."

35a ANSI BASIC, p. 99: "The value of the function NUM shall be the number of data supplied in the last array receiving data via an array-input-statement."

36 Waite and Mather, BASIC, p. 60, hier mit Anpassung von Statement 30 an die ANSI BASIC-Norm.

```
10 DIM A(100)
20 LET S = 0
30 MAT INPUT A(?)
40 FOR I=1 TO NUM
50    LET S = S + A(I)
60 NEXT I
70 PRINT "SUMME DER EINGELESENEN WERTE =";S
80 PRINT "ANZAHL DER EINGELESENEN WERTE =";NUM
90 END
```

Nicht vorgesehen ist nach ANSI BASIC die Angabe einer DIM-umfassenden Eingabeliste im MAT INPUT-Statement. Daraus folgt, dass – in Analogie zum MAT READ-Statement – die in den Eingabelisten angegebenen Felder durch vorausgehende Statements dimensioniert werden müssen.[36a]

Mit der FOSBIC Version 2 wird die ANSI BASIC-Norm vollständig realisiert. Zusätzlich sind die oben vorgetragenen Erweiterungen bezüglich des MAT READ-Statements ebenfalls eingebaut.

In konsequenter Fortsetzung der Standardisierung hat ANSI BASIC auch das MAT LINPUT-Statement in die Norm aufgenommen. Allgemein gilt:[37]

⟨MAT LINPUT-Statement 227c 209⟩ : =

MAT LINPUT ⟨MAT Linputliste 227d⟩ |
MAT LINPUT ⟨PROMPT-Identifikation 117b⟩ ⟨MAT Linput-liste 227d⟩

⟨MAT Linput-liste 227d 227c⟩ : = ⟨Alphanumerische Deklaration 92⟩ | ⟨Alphanumerische Eintragung 218⟩ |

⟨Variabler Vektor 227b⟩ | ⟨MAT Linput-liste 227d⟩ , ⟨MAT Linput-liste 227d⟩

36a Normalerweise geschieht dies durch das DIM-Statement; vgl. Bd. 1, Abs. 331.21. – Unter Umständen können jedoch auch die Initialisierungsfunktionen der MAT-Wertzuordnungsanweisungen benutzt werden; vgl. Abs. 331.22.

37 Zu früheren Entwicklungen des MAT LINPUT-Statements, insbesondere im Rahmen von NCC Standard BASIC (p. 142), vgl. Weber und Türschmann, BASIC 2, 1. Aufl., S. 65.

Der nach ANSI BASIC vorgesehene Aufbau der MAT Linputliste ist insofern noch etwas begrenzt, als nur alphanumerische Eintragungen, ohne Stringlängen-Trimmer, verwendbar sind und – im Gegensatz zur MAT Inputliste – die Benutzung des Variablen Vektors nicht vorgesehen ist.[38]

Funktionell kann das MAT LINPUT-Statement als multiple Form des elementaren LINPUT-Statements interpretiert werden.[38a] Sofern es bei einzelnen BASIC-Compilersystemen nicht verfügbar ist, kann es meistens durch entsprechende MAT INPUT-Statements ersetzt werden, wobei der einzige Nachteil darin besteht, dass die Begrenzungszeichen(") für Texte nicht mitaufgenommen werden können.

331.2 MAT-Wertzuordnungsanweisungen

Die MAT-Wertzuordnungsanweisungen sind generell dadurch charakterisiert, dass sie zu einer ganzheitlichen Matrizenverarbeitung führen. Dies im Gegensatz zur elementbezogenen Matrizenverarbeitung unter ausschliesslicher Verwendung der aus Band 1 bekannten Elementaranweisungen.[39]

Grundsätzlich lassen sich zwei Typen von MAT-Wertzuordnungsanweisungen (mat-assignment statements) unterscheiden:

$$\underset{228}{\left\langle \begin{array}{l}\text{MAT-Wertzuordnungs-}\\ \text{anweisung}\end{array}\right\rangle} := \underset{208}{\left\langle \begin{array}{l}\text{MAT LET-}\\ \text{Statement}\end{array}\right\rangle} \;\Big|\; \underset{229}{\left\langle \begin{array}{l}\text{MAT let-}\\ \text{Statement}\end{array}\right\rangle} \quad \underset{237}{}$$

38 ANSI BASIC, p. 99: 'Execution of an array-linput-statement shall assign the entire contents of successive linput-replies (excluding their end-of-lines) in row order to elements of the string-arrays in the string-array-list. The number of linput-replies requested shall equal the number of elements requiring values. If a redim is present then dynamic redimensioning shall take place before values are assigned to the redimensioned array."

38a Vgl. Bd. 1, S. 169.

39 Vgl. hierzu auch NCC Standard BASIC, p. 126: "As in most programming languages, matrices and vectors can be manipulated via their elements. It is, however, often more convenient to regard matrices and vectors as entities rather than as indexed collections of entities, and to manipulate them in their entirety." – Ähnlich auch, mit sofortiger Differenzierung zwischen numerischen und alphanumerischen MAT-Wertzuordnungsanweisungen, ANSI BASIC, p. 48, 52: "As in most programming languages, numeric-arrays in BASIC can be manipulated by manipulating their elements. However, it is often more convenient to regard numeric-arrays as entities rather than as in-

Diese Differenzierung entspricht der früher vorgenommenen Unterscheidung zwischen LET- und let-Startements.[40]
Nach ANSI BASIC wird nur die Form ohne LET-Kennwort normiert.[40a]
Aus Gründen der Kontinuität und der didaktischen Darstellungsweise wird weiterhin die bereits in der 1. Auflage vorgenommene Unterscheidung in zwei MAT-Wertzuordnungsanweisungen beibehalten, manchmal aber bereits bei Behandlung des MAT LET-Statements auf einschlägige Normvorschläge von ANSI BASIC verwiesen.

331.21 MAT LET-Statements

Die MAT LET-Statements lassen sich primär – in Analogie zu den einfachen LET-Statements[41] – in numerisches und alphanumerische MAT LET-Statements einteilen, so dass sich allgemein formuliert:

$$\underset{229}{\left\langle \begin{matrix}\text{MAT LET-}\\ \text{Statement}\end{matrix} \right\rangle} \; := \; \underset{230}{\left\langle \begin{matrix}\text{Numerisches}\\ \text{MAT LET-}\\ \text{Statement}\end{matrix} \right\rangle} \;\Big|\; \underset{234}{\left\langle \begin{matrix}\text{Alphanumerisches}\\ \text{MAT LET-}\\ \text{Statement}\end{matrix} \right\rangle}$$

331.211 Numerisches MA LET-Statement
Für die numerischen MAT LET-Statements gilt generell

$$\underset{230}{\langle \text{Numerisches MAT LET-Statement} \rangle} \; := \; \underset{229}{}$$

$$\boxed{\text{MAT LET} \; \underset{231a}{\left\langle \begin{matrix}\text{Numerisches}\\ \text{Zielfeld}\end{matrix} \right\rangle} \; = \; \underset{232}{\left\langle \begin{matrix}\text{Numerischer}\\ \text{MAT-Ausdruck}\end{matrix} \right\rangle}}$$

Die vorstehende metalinguistische Gleichung besagt, dass bei Ausführung eines numerischen MAT LET-Statements der Wert des rechtsstehenden MAT-Ausdrucks dem links des Zuordnungszeichens angegebe-

dexed collections of entities, and to manipulate the entire entity at once. BASIC provides a number of standard operations to facilitate such manipulations. ...As with numeric-arrays, string-arrays may be regarded as entities rather than as indexed collections of entities. BASIC provides the ability to concatenate and assign entire arrays of strings."

40 Vgl. Bd. 1, Abs. 332.3.
40a ANSI BASIC, p. 48–53.
41 Vgl. Bd. 1, Abs. 332.311 und Abs. 332.312.

nen Zielfeld (array destination) zuzuweisen ist.[42] Die Wertzuweisung erfolgt für alle Elemente des Zielfeldes, mit Ausnahme von Nullzeile und Nullspalte (bei Felddimensionierung auf der Basis 0).[43]
Das Zielfeld wird normalerweise durch einen Feldnamen bezeichnet; in Ausnahmefällen kann auch eine einfache numerische Variable als Zielfeld in Erscheinung treten.[44] Dies entspricht den Regeln der Matrixalgebra, die auch nach ANSI BASIC für numerische MAT-Wertzuordnungsanweisungen einzuhalten sind.[44a]
Für das numerische Zielfeld gilt mithin:

$$\begin{matrix}\langle\text{Numerisches}\\ \text{Zielfeld}\rangle\\ 231\quad 230\end{matrix} := \langle\text{Feldname}\rangle_{40} \mid \begin{matrix}\langle\text{Einfache}\\ \text{numerische Variable}\rangle\\ 37\end{matrix}$$

Nach den bei ANSI BASIC angegebenen metalinguistischen Gleichungen kann auf der linken Seite einer numerischen MAT-Wertzuordnungsanweisung keine einfache Variable erscheinen.[44b] Diese Einschränkung findet sich bei nahezu allen BASIC-Prozessoren.[44c] Eine einfache numerische Variable ist diesfalls durch eine Matrix mit der Dimension (1) oder (1,1) zu ersetzen.[44d] Damit gilt für das numerische Zielfeld nur noch:

$$\underset{231a}{\langle\text{Numerisches Zielfeld}\rangle} \underset{230}{:=} \underset{40}{\langle\text{Feldname}\rangle}$$

Um die enge Verbindung zwischen Elementar- und Matrizenanweisungen in der Programmiersprache BASIC zu festigen, sollte die Form ge-

42 NCC Standard BASIC, p. 130: "Execution of a mat-assignment statement causes the (elements of the) array on the left of the equal sign to be assigned a new value."

43 NCC Standard BASIC, p. 130: "In all cases the elements in row and column zero are not affected by a mat-assignment statement." – Zur Felddimensionierung auf Basis Null vgl. Bd. 1. Abs. 331.211.

44 In diesem Sinne stellte auch das ANSI X3J2 Matrix Subcommittee fest (X3J2/75-23, G-10): "The... MAT operations cause the output matrix to have the form of a doubly subscripted array, except that if the result is a 1 by 1 array, then the result may be stored into a numeric-variable."

44a ANSI BASIC, p. 49: "The evaluation of numeric-array-expressions shall follow the normal rules of matrix algebra."

44b ANSI BASIC, p. 48: "numeric-array-assignment = MAT numeric-array-name equals-sign numeric-array-expression."

44c Control Data, BASIC Version 3. Reference Manual, p. 7.2; Nixdorf, System-Software Business-Basic, S. 10.19, 10.40–43; Sperry Rand, Sperry UNIVAC 1100 Series. BASIC, p. 4.20–21.

44d Vgl. Karl Weber. BASIC für Anfänger. Bern 1979, S. 121.

mäss Gleichung 231 als Norm diskutiert werden; sie wird vom FOSBIC-Compiler eingehalten. Ausserdem wird die Matrizenalgebra dadurch allgemeiner realisiert, da eine Matrizenoperation im Ergebnis zu einem Skalar führen kann.
Sinnvoll ist auch die Zulassung multipler MAT LET-Statements, insbesondere in Verbindung mit MAT-Initialisierungsfunktionen; vgl. Abs. 331.221. Damit würde auch die bei den elementaren Werzuordnungsanweisungen bereits standardisierte Form konsequent in den Matrizenanweisungen fortgesetzt; vgl. Bd. 1, Abs. 331.31.
Nach ANSI BASIC wird verlangt, dass das numerische Zielfeld in seinen Feldgrössen den Werten des numerischen MAT-Ausdrucks dynamisch angepasst wird.[44e] Die neuen Feldgrössen müssen nicht grundsätzlich kleiner oder gleich den ursprünglich dimensionierten Werten sein, sondern können diese auch überschreiten, falls die Gesamtzahl der Elemente des redimensionierten Feldes nicht die Gesamtzahl der Ausgangsdimensionierung übersteigt.[44f] Mithin gelten auch hier die wertmässigen Grenzen, die explizit in Gleichung 220 und 221 für die DIM-begrenzten Zeilen- und Spaltenzahlen festgelegt wurden.[44g] Im Gegensatz zum DIM-begrenzten MAT READ- und MAT INPUT-Statement bleiben die via MAT LET-Statement neu festgelegten Feldgrenzen im nachfolgenden Programmablauf so lange bestehen, bis eventuell eine erneute Redimensionierung erfolgt. Vgl. Abschnitt 331.22 für ein ausführliches Beispiel.
Der in der metalinguistischen Gleichung 230 erscheinende numerische MAT-Ausdruck bestimmt den Charakter des MAT LET-Statements. Generell gilt:

44e ANSI BASIC, p. 49: "Execution of a numeric-array-assignment shall cause the numeric-array-expression to be evaluated and its value assigned to the array named to the left of the equals-sign. If appropriate, this array shall be redimensioned dynamically; i. e., its number of dimensions shall be unchanged, but its size shall be changed to conform to the size of the array which is the value of the numeric-array-expression."

44f ANSI BASIC, p. 49: "When an numeric-array is redimensioned dynamically, the current upper bounds for its subscripts shall be changed to conform to the size of its new value and the current lower bounds shall be retained. The new upper bounds need not be individually less than or equal to the bounds specified in the dimension-statement for that numeric-arry (or by the default dimensions if the numeric-arry is not dimensioned explicitly), as long as the total number of elements for the redimensioned numeric-array does not exceed the total number of elements specified by the array's original dimensions."

44g Vgl. Abs. 331.112.

⟨Numerischer MAT-Ausdruck⟩ 232 230 : = (⟨Arithmetischer Ausdruck⟩ 49) | ⟨Feldname⟩ 40 |

(⟨Arithmetischer Ausdruck⟩ 49) ⟨Dyadischer Operator⟩ 12 ⟨Feldname⟩ 40 |

⟨Feldname⟩ 40 ⟨Dyadischer Operator⟩ 12 (⟨Arithmetischer Ausdruck⟩ 49) |

⟨Feldname⟩ 40 ⟨Dyadischer MAT-Operator⟩ 233 ⟨Feldname⟩ 40 |

⟨Arithmetische MAT-Funktion⟩ 239 |

(⟨Arithmetischer Ausdruck⟩ 49) ⟨Dyadischer Operator⟩ 12 ⟨MAT-Initialisierungsfunktion⟩ 240 |

⟨MAT-Initialisierungsfunktion⟩ 240 ⟨Dyadischer Operator⟩ 12 (⟨Arithmetischer Ausdruck⟩ 49)

Als dyadische MAT-Operatoren treten nur die in Gleichung 233 angegebenen Zeichen auf; die bei den dyadischen Operatoren möglichen Zeichen / (Division) und ** bzw. ↑ (Exponentiation) sind nicht zulässig.

⟨Dyadischer MAT-Operator⟩ 233 : = + | − | * 232

Danach können grundsätzlich sechs verschiedene Typen von MAT LET-Statements unterschieden werden.
Zunächst besteht die Möglichkeit, mit Hilfe des MAT LET-Statements den Wert eines arithmetischen Ausdrucks allen Elementen des Zielfeldes zuzuweisen. Beispiele zur Verarbeitung des Statementtyps

$$\text{MAT LET} \begin{pmatrix}\text{Numerisches}\\ \text{Zielfeld}\end{pmatrix} = (\langle\text{Arithmetischer Ausdruck}\rangle)$$

sind aus den folgenden Programmen ersichtlich. Für Kontrollzwecke wurden jeweils die erzeugten Resultate mit Hilfe entsprechender MAT PRINT-Statements ausgedruckt.[45] Die vermittelten Druckbilder können beim FOSBIC-Compiler durch die Angabe

```
* HEADLINE
```

in der ersten Karte eines BASIC-Jobs angesteuert werden, wodurch jede Matrizenausgabe genau gekennzeichnet und der Anfänger damit bei der Ausgabe und Kontrolle seiner Matrizenprogramme unterstützt wird.
Das Programm

```
10 DIM A(10)
20 MAT LET A = (5)
30 MAT PRINT A
40 END
```

bewirkt die Zuordnung einer arithmetischen Konstanten an alle Elemente des Vektors A, wie auch aus dem Druckbild

```
VEKTOR *A*
   5            5            5            5
   5            5            5            5
```

ersichtlich ist.
Mit dem Programm

```
10 DIM B(3,4)
20 LET A1 = 10
30 MAT LET B = (A1)
40 MAT PRINT B
50 END
```

wird der Matrix B der Wert der Variablen A1 zugewiesen, mit dem Ergebnis.

```
MATRIX *B* ZEILE *   1*
   10           10           10           10
MATRIX *B* ZEILE *   2*
   10           10           10           10
MATRIX *B* ZEILE *   3*
   10           10           10           10
```

Dass der numerische MAT-Ausdruck auch komplexere Strukturen aufweisen kann, zeigt das folgende Programm.

45 Vgl. Abs. 331.31.

```
10 DIM A(2,3)
20 LET X = 5
30 READ Y
40 MAT LET A = (X*SQR(Y)-3)
50 MAT PRINT A
60 DATA 9
70 END
```

Die zweite Form des MAT LET-Statements erlaubt die Übertragung einer Matrix an ein anderes, gleich-, kleiner- oder grösserdimensioniertes Zielfeld.[46] Dieser Operation entspricht das Statement

MAT LET ⟨Numerisches Zielfeld⟩ = ⟨Feldname⟩

Vgl. hierzu das Programm

```
10 REM WIRKUNG DER UNTERSCHIEDLICHEN DIMENSIONIERUNG BEACHTEN
20 DIM A(4,5), B(2,10), C(3,5), D(2,4)
30 MAT LET B = (5)
40 FOR I=1 TO 10
50   LET B(2,I)=2.33
60 NEXT I
70 MAT LET D = (8)
80 MAT LET C = (1)
90 MAT LET A = B
100 MAT LET C = B
110 MAT PRINT A, B
130 MAT PRINT C, D
140 END
```

mit der Ergebnisliste

```
MATRIX *A* ZEILE *   1*
   5              5              5              5              5
MATRIX *A* ZEILE *   2*
   2.33           2.33           2.33           2.33           2.33
MATRIX *A* ZEILE *   3*
   0.0            0.0            0.0            0.0            0.0
MATRIX *A* ZEILE *   4*
   0.0            0.0            0.0            0.0            0.0
MATRIX *B* ZEILE *   1*
   5              5              5              5              5
   5              5              5              5              5
MATRIX *B* ZEILE *   2*
   2.33           2.33           2.33           2.33           2.33
   2.33           2.33           2.33           2.33           2.33
MATRIX *C* ZEILE *   1*
   5              5              5              5              5
MATRIX *C* ZEILE *   2*
   2.33           2.33           2.33           2.33           2.33
MATRIX *C* ZEILE *   3*
   1              1              1              1              1
MATRIX *D* ZEILE *   1*
   8              8              8              8
MATRIX *D* ZEILE *   2*
   8              8              8              8
```

Zielfeld und zu übertragendes Feld können in ihrer Zeilen- und Spaltendimensionierung unterschiedlich aufgebaut sein. Der FOSBIC-Compiler nimmt als Übertragungsgrenzen die jeweils kleineren Grenzen an, so dass

46 NCC Standard BASIC, p. 130: "That mat-assignment statement provides for the assignment of an arithmetic... array value to an array destination."

keine Überspeicherung in benachbarte Felder zugelassen wird, aber eine Teilübertragung von grösseren Feldern an kleinere oder umgekehrt möglich ist; vgl. hierzu obiges Beispiel.
Die dritte Form des MAT LET-Statements bezieht sich auf die arithmetische Verknüpfung einer Matrix mit einem Skalar, dargestellt durch

$$\text{MAT LET} \left\langle \begin{matrix}\text{Numerisches}\\ \text{Zielfeld}\end{matrix} \right\rangle = \left(\left\langle \begin{matrix}\text{Arith-}\\ \text{metischer}\\ \text{Ausdruck}\end{matrix} \right\rangle \right) \left\langle \begin{matrix}\text{Dyadi-}\\ \text{scher}\\ \text{Operator}\end{matrix} \right\rangle \left\langle \begin{matrix}\text{Feld-}\\ \text{name}\end{matrix} \right\rangle$$

und

$$\text{MAT LET} \left\langle \begin{matrix}\text{Numerisches}\\ \text{Zielfeld}\end{matrix} \right\rangle = \left\langle \begin{matrix}\text{Feld-}\\ \text{name}\end{matrix} \right\rangle \left\langle \begin{matrix}\text{Dyadi-}\\ \text{scher}\\ \text{Operator}\end{matrix} \right\rangle \left(\left\langle \begin{matrix}\text{Arithme-}\\ \text{tischer}\\ \text{Ausdruck}\end{matrix} \right\rangle \right)$$

ANSI BASIC erlaubt nur die erste Form der Verknüpfung eines arithmetischen Ausdrucks mit einer Matrix, wobei nur die Multiplikation zugelassen ist.[46a] Der FOSBIC-Compiler lässt alle nach der Matrizenalgebra denkbaren Formen zu, wobei die in der Matrizenalgebra festgelegte Bearbeitungsweise des Matrizenausdrucks von links nach rechts streng beachtet wird. So ist beispielsweise deutlich zwischen den Statements

```
230  MAT LET X=(10)/Y
```

und

```
230 MAT LET X=Y/(10)
```

zu unterscheiden, da im ersten Fall die Skalare 10 durch jedes Element von Y dividiert und erst dann das Ergebnis dem jeweiligen Element von X zugewiesen wird. Im zweiten Fall wird jedes Element von Y durch 10 dividiert und in X gespeichert.
Die Einsatzmöglichkeiten dieses Statementtyps sind aus den folgenden, mit Hilfe von REM-Statements kommentierten, Beispielen ersichtlich. Die Drucklisten sollen dem Leser eine rasche Operationskontrolle ermöglichen.

46a ANSI Basic, p. 48–49. Die metalinguistische Darstellung widerspricht der Erläuterung im Text, die besagt: "When a scalar-multiplier is present in a numeric-array-expression, the numeric-expression shall be evaluated, and then each element of the numeric-array shall be multiplied by this value."

```
10 REM ADDITION
20 REM MATRIX = (ARITHMETISCHER AUSDRUCK) + MATRIX
30 DIM A(5)
40 MAT INPUT A
50 MAT LET A = (5) + A
60 MAT PRINT A
70 END LIST DATA

LIST OF DATA AFTER END STATEMENT

1,2,3,4,5
*

VEKTOR *A*
   6          7          8          9          10
```

```
10 REM SUBTRAKTION
20 REM MATRIX = MATRIX - (ARITHMETISCHER AUSDRUCK)
22 REM                   ODER
25 REM MATRIX = (ARITHMETISCHER AUSDRUCK) - MATRIX
30 DIM A(3), B(5), C(7)
40 MAT READ A
50 MAT LET B = A - (SIN(0.7854))
55 MAT LET C = (COS(A(2))) - A
60 MAT PRINT A, B, C
70 REM WIRKUNG UNTERSCHIEDLICH DIMENSIONIERTER
80 REM MATRIZEN BEACHTEN
85 DATA 1,2,3
90 END

VEKTOR *A*
   1          2          3
VEKTOR *B*
  0.292892    1.29289    2.29289    0.0        0.0
VEKTOR *C*
 -1.41615    -2.41615   -3.41615    0.0        0.0
  0.0         0.0
```

```
10 REM MULTIPLIKATION
20 REM MATRIX = MATRIX * (ARITHMETISCHER AUSDRUCK)
30 DIM A(3,4), B(3,4)
40 READ C, D
50 MAT READ B
60 MAT LET A = B*(C**D)
70 MAT PRINT A
80 DATA 2, 3, 1,2,3,4,5,6,7,8,9,10,11,12
90 END

MATRIX *A* ZEILE *   1*
  8          16         24         32
MATRIX *A* ZEILE *   2*
 40          48         56         64
MATRIX *A* ZEILE *   3*
 72          80         88         96
```

```
10 REM DIVISION
20 REM MATRIX = MATRIX / (ARITHMETISCHER AUSDRUCK)
22 REM                   ODER
25 REM MATRIX = (ARITHMETISCHER AUSDRUCK) / MATRIX
30 DIM A(5), B(5), C(5)
40 MAT READ A
50 MAT LET B = A/(SIN(0.7854))
55 MAT LET C = (10)/A
60 MAT PRINT A, B, C
70 DATA 1,2,3,4,5
80 END

VEKTOR *A*
   1          2          3          4          5
VEKTOR *B*
  1.41421     2.82842    4.24263    5.65684    7.07105
VEKTOR *C*
 10           5          3.33333    2.5        2
```

```
10 REM EXPONENTIATION
20 REM MATRIX = MATRIX ** (ARITHMETISCHER AUSDRUCK)
22 REM                    ODER
25 REM MATRIX = (ARITHMETISCHER AUSDRUCK) ** MATRIX
27 DIM S(2,5), T(2,5)
30 MAT INPUT R(2,5)
40 MAT LET S = R**(3)
45 MAT LET T = (2)**R
50 MAT PRINT R, S, T
60 END LIST DATA

LIST OF DATA AFTER END-STATEMENT

 10,20,30,40,50,50,60,70,80,90
 *

MATRIX *R* ZEILE *   1*
   10          20          30          40          50
MATRIX *R* ZEILE *   2*
   50          60          70          80          90
MATRIX *S* ZEILE *   1*
 1000        8000       27000       64000      125000
MATRIX *S* ZEILE *   2*
125000      216000      343000      512000      729000
MATRIX *T* ZEILE *   1*
 1024        1048580     1073740000  1099518000000  1.12590E+15
MATRIX *T* ZEILE *   2*
1.12590E+15  1.15292E+18  1.18059E+21  1.20893E+24  1.23794E+27
```

Für den vierten Typ des MAT LET-Statements gilt generell

$$\text{MAT LET} \left\langle \begin{matrix}\text{Numerisches}\\ \text{Zielfeld}\end{matrix} \right\rangle = \left\langle \begin{matrix}\text{Feld-}\\ \text{name}\end{matrix} \right\rangle \left\langle \begin{matrix}\text{Dyadischer}\\ \text{MAT-Operator}\end{matrix} \right\rangle \left\langle \begin{matrix}\text{Feld-}\\ \text{name}\end{matrix} \right\rangle$$

Als Operationszeichen sind diesfalls nur die dyadischen MAT-Operatoren zugelassen, so dass eine Matrizenverknüpfung nur via Addition, Subtraktion oder Multiplikation erfolgen kann. Dabei sind für die Matrizenoperationen generell gültigen Regeln einzuhalten.[47] So können nur Matrizen gleicher Ordnung addiert oder subtrahiert werden. Weiterhin ist die Multiplikation der Matrizen A und B in der Reihenfolge AB = C nur durchführbar, wenn die Matrix A so viele Spalten besitzt, wie die Matrix B Zeilen hat; die resultierende Matrix C hat so viele Zeilen wie die Matrix A und so viele Spalten wie die Matrix B, mithin:

$$C_{mp} = A_{mn}B_{np}$$

47 Vgl. Heinz Körth, Carl Otto, Walter Runge und Manfred Schoch. Lehrbuch der Mathematik für Wirtschaftswissenschaften. Opladen 1972. S. 156–178; Rudolf Zurmühl. Matrizen und ihre technischen Anwendungen. 4. Aufl., Berlin 1964.

Darauf ist bei der Matrizendimensionierung bzw. -Redimensionierung sorgfältig zu achten.[48]

Während beim FOSBIC-Compiler die Matrizenmultiplikation der Form

$$A_{mn} = A_{mn}B_{nn}$$

und

$$A_{mn} = B_{mm}A_{mn}$$

möglich ist, ist die Form

$$A_{nn} = A_{nn}A_{nn}$$

nicht zulässig. Der FOSBIC-Compiler Version 2 verarbeitet alle drei Formen der Matrizenmultiplikation korrekt. Die Mehrzahl der BASIC-Prozessoren[48a] verbietet die gleichzeitige Verwendung desselben Matrizennamens auf der linken und rechten Seite einer MAT-Wertzuordnungsanweisung, obwohl dies nach ANSI BASIC noch normgerecht ist.[48b] Zeilenvektoren müssen in diesem Fall als einzeilige Matrizen dimensioniert sein, vgl. folgendes Beispiel (mit MAT-Initialisierungsfunktion ZER gemäss Ausführungen auf S. 95–96):

```
10 DIM A(10,1), B(10,10)
20 READ N
30 MAT LET A=ZER(1,N)
40 MAT LET B=ZER(N,N)
50 MAT READ A,B
60 MAT LET A=A*B
70 MAT PRINT A
80 DATA 4
90 DATA 1,2,3,4
95 DATA 2,2,2,2, 3,3,3,3, 4,4,4,4, 5,5,5,5
99 END

MATRIX *A* ZEILE *   1*
   40          40          40          40
```

Die einzelnen Operationen werden im folgenden anhand einfacher Beispiele dargestellt; die beigegebenen Drucklisten sollen dem Leser wiederum die Rechenkontrolle ermöglichen.

48 NCC Standard BASIC, p. 130–131: "The mat-assignment statement... provides for the addition (+) subtraction (−) and multiplication (*) of matrices and vectors... and the scalar multiplication of two vectors. The dimensions of the arrays in a mat-assignment statement must conform with the usual rules of matrix algebra. If a vector is multiplied by a matrix, it is treated as a row or column vector as appropriate; the result, however, must be assigned to a vector destination. If two vectors are multiplied together the result is a scalar (the inner product) and must be assigned to a scalar destination."

48a ANSI BASIC, p. 48–49.

48b Beispielsweise Control Data, BASIC Version 3. Reference Manual, p. 7.2; Sperry Rand, Sperry UNIVAC 1100 Series. BASIC, p. 4.21.

```
10 REM MATRIZENADDITION
20 REM MN-MATRIX = MN-MATRIX + MN-MATRIX
30 DIM A(2,3), B(2,3), C(2,3)
40 MAT READ A, B
50 MAT LET C = A + B
60 MAT PRINT A, B, C
70 DATA 1,2,3,4,5,6,7,8,9,10,11,12
80 END
MATRIX *A* ZEILE *   1*
   1              2              3
MATRIX *A* ZEILE *   2*
   4              5              6
MATRIX *B* ZEILE *   1*
   7              8              9
MATRIX *B* ZEILE *   2*
  10             11             12
MATRIX *C* ZEILE *   1*
   8             10             12
MATRIX *C* ZEILE *   2*
  14             16             18
```

```
10 REM MATRIZENSUBTRAKTION
20 REM MN-MATRIX = MN-MATRIX - MN-MATRIX
30 DIM A(2,3), B(2,3), C(2,3)
40 MAT INPUT C, B
50 MAT LET A = C - B
60 MAT PRINT A
70 END LIST DATA

LIST OF DATA AFTER END STATEMENT

8, 10, 12, 14, 16, 18, 7, 8, 9, 10, 11, 12

MATRIX *A* ZEILE *   1*
   1              2              3
MATRIX *A* ZEILE *   2*
   4              5              6
```

```
10 REM MATRIZENMULTIPLIKATION
20 REM MP-MATRIX = MN-MATRIX * NP-MATRIX
30 DIM A(2,3), B(3,2), C(2,2), D(3,3)
40 MAT INPUT A, B
50 MAT LET C = A*B
60 MAT PRINT C
70 MAT LET D = B*A
80 MAT PRINT D
90 END LIST DATA

LIST OF DATA AFTER END STATEMENT

2, -1, 0, 3, 1, -2, 1, 4, -2, 0, 1, 3
*

MATRIX *C* ZEILE *   1*
   4              8
MATRIX *C* ZEILE *   2*
  -1              6
MATRIX *D* ZEILE *   1*
  14              3             -8
MATRIX *D* ZEILE *   2*
  -4              2              0.0
MATRIX *D* ZEILE *   3*
  11              2             -6
```

Das letzte Beispiel führte zur Einlesung der Matrizen

$$A_{2\times 3} = \begin{bmatrix} 2 & -1 & 0 \\ 3 & 1 & -2 \end{bmatrix} \quad \text{und } B_{3\times 2} = \begin{bmatrix} 1 & 4 \\ -2 & 0 \\ 1 & 3 \end{bmatrix}$$

und zur Berechnung von

$$C_{2\times 2} = AB = \begin{bmatrix} 4 & 8 \\ -1 & 6 \end{bmatrix} \quad \text{sowie } D_{3\times 3} = BA = \begin{bmatrix} 14 & 3 & -8 \\ -4 & 2 & 0 \\ 11 & 2 & -6 \end{bmatrix}$$

Sonderfälle von Matrizenprodukten sind das skalare und dyadische Produkt zweier Vektoren.
Ein Skalarprodukt fällt bei Multiplikation eines Zeilenvektors mit einem Spaltenvektor von je n Komponenten an. Umgekehrt ergibt die Multiplikation eines n Elemente umfassenden Spaltenvektors mit einem m Elemente umfassenden Zeilenvektor eine nm-Matrix. Vgl. hierzu die folgenden Beispiele, bei denen insbesondere auch die Angaben zur Dimensionierung zu beachten sind.

```
10 REM SKALARES PRODUKT ZWEIER VEKTOREN
20 REM SKALAR = 1N-MATRIX * N1-MATRIX
30 REM FELD FUER SKALAR MUSS NICHT DIMENSIONIERT WERDEN
40 DIM A(3), B(3,1)
50 MAT INPUT A, B
60 MAT LET C = A*B
70 MAT PRINT A, B
80 PRINT "ERGEBNIS =", C
90 END LIST DATA

LIST OF DATA AFTER END STATEMENT

1, -3, 2, 2, 4, -1
*

VEKTOR "A"
  1          -3          2
MATRIX "B" ZEILE "  1"
  2
MATRIX "B" ZEILE "  2"
  4
MATRIX "B" ZEILE "  3"
 -1
ERGEBNIS =          -12
```

Um der ANSI BASIC-Norm zu entsprechen, müsste das vorige Beispiel wie folgt umgeformt werden:

```
10 REM SKALARES PRODUKT ZWEIER VEKTOREN
20 REM SKALAR = ZEILENVEKTOR * SPALTENVEKTOR(N*1 MATRIX)
30 REM FELD FUER SKALARWERT MUSS NACH ANSI BASIC DIMENSIONIERT SEIN
40 DIM A(3), B(3,1), C(1)
50 MAT INPUT A, B
60 MAT LET C = A*B
70 MAT PRINT A, B
80 PRINT "SKALARWERT=";C(1)
90 END
```

Das Beispiel zum dyadischen Vektorprodukt ist normkonform, wird allerdings von den meisten BASIC-Prozessoren nur verarbeitet, wenn alle Matrizen in der Multiplikation mit zwei Indizes dimensioniert wurden.

```
10 REM DYADISCHES PRODUKT ZWEIER VEKTOREN
20 REM NN-MATRIX = N1-MATRIX * 1N-MATRIX
30 REM DIMENSIONIERUNG BEACHTEN
40 DIM A(3,1), B(3), C(3,3)
50 REM DIE DIMENSIONIERUNG KANN AUCH LAUTEN
60 REM A(3,1), B(1,3), C(3,3)
70 MAT READ A, B
80 MAT LET C = A*B
90 MAT PRINT A, B, C
95 DATA 1, -3, 2, 2, 4, -1
99 END
```

```
MATRIX "A" ZEILE "   1"
   1
MATRIX "A" ZEILE "   2"
  -3
MATRIX "A" ZEILE "   3"
   2
VEKTOR "B"
   2              4              -1
MATRIX "C" ZEILE "   1"
   2              4              -1
MATRIX "C" ZEILE "   2"
  -6             -12               3
MATRIX "C" ZEILE "   3"
   4              8              -2
```

Bezüglich der MAT LET-Statements vom Typ

$$\text{MAT LET} \begin{pmatrix}\text{Numerisches}\\ \text{Zielfeld}\end{pmatrix} = \begin{pmatrix}\text{Arithmetische}\\ \text{MAT-Funktion}\end{pmatrix}$$

sei auf die entsprechenden Ausführungen in Abschnitt 331.22 verwiesen. Bei der Programmkompilierung werden vom FOSBIC-Compiler folgende MAT LET-Fehler erkannt:

– Kennwortfehler bei MAT:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

– Kennwortfehler bei LET:

```
        I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B    331
```

– Fehlerhafter Feldname:

```
        ILLEGAL FIELD NAME IN THE MAT STATEMENT ABOVE "0"
SEE BASIC TEXTBOOK  B 331.21
```

– Fehlendes Zuordnungszeichen:

```
        I CANNOT FIND AN EQUAL SIGN IN THE MAT LET STATEMENT ABOVE
SEE BASIC TEXTBOOK  B 331.21
```

- Linksseitiger Fehler im MAT LET-Statement:

```
ERROR ON THE LEFT SIDE OF A MAT LET STATEMENT
I DO NOT UNDERSTAND THE FOLLOWING PART IN THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B 331.21
```

- Unzulässige Matrizenmultiplikation:

```
NO ACCEPTABLE MATRIX MULTIPLICATION
I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B331.211
```

- Inadäquate Felder:

```
VARIABLE "0" IS DIMENSIONED AS A VECTOR --
VARIABLE "0" IS DIMENSIONED AS A MATRIX
I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B331.211
```

- Fehlende Dimensionierung:

```
VARIABLE "0" IS NOT DIMENSIONED
I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B331.111
```

- Nicht korrespondierende Dimensionierung:

```
VARIABLE "0" IS DIMENSIONED AS A VECTOR BUT YOU TREAT IT AS A MATRIX
I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B 331.21
```

- Falsche Dimensionierung durch arithmetischen Ausdruck:

```
DIMENSION NOT ALLOWED WITH EXPRESSION
I DO NOT UNDERSTAND THE FOLLOWING PART IN THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B331.211
```

- Allgemeiner Fehler im MAT LET-Statement:

```
ILLEGAL MAT LET STATEMENT
SEE BASIC TEXTBOOK  B331.211
I DO NOT LIKE THE FOLLOWING CHARACTER IN THE MAT STATEMENT ABOVE "0"
SEE BASIC TEXTBOOK  B331.211
```

- Elementarfehler im arithmetischen Teil des MAT-Ausdrucks: Vgl. Bd. 1, Abs. 324.1

Prozessor →	ANSI BASIC 79	Dartmouth BASIC-6	UOM BASIC/ UBASIC	IBM CALL BASIC	FOSBIC	Wang Labo-ratories	Nixdorf	Digital Equipment
MAT mit oder ohne Kennwort LET	ohne	ohne	ohne	ohne	mit/ohne	ohne	ohne	mit/ohne
numerisch	×	×	×	×	×	×	×	×
alphanumerisch	×	×	×		×	×	×	
MAT-Funktionen								
ZER	×	×	×	×	×	×	×	×
CON	×	×	×	×	×	×	×	×
IDN	×	×	×	×	×	×	×	×
TRN	×	×	×	×	×	×	×	
INV	×	×	×	×	×	×	×	
DET	×	×				×		
NUL	×	×						
Dimensionierung in MAT-Initialisierungsfunktion	ja	nein	ja	nein	ja	ja	ja	ja
Dimensionierungsart	arithm. Ausdruck		Ganzzahl		Ganzzahl	arithm. Ausdruck	arithm. Ausdruck	Ganzzahl
Redimensionier. in MAT-Initialisierungsfunktionen	ja	ja, nur CON/IDN	nein	ja	ja	ja	ja	ja
Redimensionierungsart	arithm. Ausdruck	arithm. Ausdruck		arithm. Ausdruck	arithm. Ausdruck	arithm. Ausdruck	arithm. Ausdruck	

Bei der Programmausführung werden folgende Fehler moniert:

- Unkompatible Felddimensionierung:

```
PROGRAM STOPPED -- SIZE ERROR IN SUBSCRIPTS OF THE VARIABLES *  ROWS  * COLUMNS
                                                        A        3        3
                                                        B        4        5
                                                        C        2        3
```

- Fehler in Matrizenmultiplikation, mit vorausgehender Meldung über die verwendeten Feldgrenzen:

```
PROGRAM STOPPED -- ERROR IN MATRIX MULTIPLICATION
                   ERROR CODE= 30
              ***** SEE BASIC TEXTBOOK =  B331.211  *****
              ***** I WAS AT LINE NUMBER    0
```

- Unzulässige Matrizenmultiplikation:

```
PROGRAM STOPPED -- MATRIX MULTIPLICATION OF THE FORM X=X*Y NOT ALLOWED -
                   X MUST BE DIMENSIONED AS A ONE-ROW MATRIX
                   ERROR CODE= 51
              ***** SEE BASIC TEXTBOOK =  B331.211  *****
              ***** I WAS AT LINE NUMBER    0
```

- Nicht korrespondierende Feldgrenzen bei Matrizenaddition oder -subtraktion:

```
PROGRAM STOPPED -- ERROR IN MATRIX OPERATION -- ADDITION OR SUBTRACTION
                   ERROR CODE= 36
              ***** SEE BASIC TEXTBOOK =  B331.221  *****
              ***** I WAS AT LINE NUMBER    0
```

- Unzulässiger Divisorwert:

```
PROGRAM STOPPED -- DIVISION ERROR IN MAT LET STATEMENT
                   THERE IS AN ATTEMPT TO DIVIDE BY ZERO
                   ERROR CODE= 42
              ***** SEE BASIC TEXTBOOK =  B331.211  *****
              ***** I WAS AT LINE NUMBER    0
```

- Unzulässige Exponentiation:

```
PROGRAM STOPPED -- EXPONENTIATION ERROR IN MAT LET STATEMENT
                   ILLEGAL EXPONENTIATION -- THE ARGUMENT IS NEGATIVE
                   0.39560357E+34 THE EXPONENT IS  0.12345000E+05
                   ERROR CODE= 43
              ***** SEE BASIC TEXTBOOK =  B331.211  *****
              ***** I WAS AT LINE NUMBER    0
```

- Exponent über- oder unterschreitet maximal zulässigen Wert bei Exponentation:

```
PROGRAM STOPPED -- EXPONENTIATION ERROR IN MAT LET STATEMENT
                   EXPONENT OVERFLOW IN EXPONENTIATION  MAXIMUM = 99
                   ERROR CODE= 44
              ***** SEE BASIC TEXTBOOK =  B331.211  *****
              ***** I WAS AT LINE NUMBER    0
```

– Exponent über- oder unterschreitet maximal zulässigen Wert bei Multiplikation:

```
PROGRAM STOPPED -- MULTIPLICATION ERROR IN MAT LET STATEMENT
                   EXPONENT OVERFLOW OR UNDERFLOW IN MULTIPLICATION
                   0.395603570E+34 *  0.123450000E+05  MAXIMUM = 99
                   ERROR CODE= 45
              ***** SEE BASIC TEXTBOOK =  B331.211  *****
              ***** I WAS AT LINE NUMBER    0
```

331.212 Alphanumerisches MAT LET-Statement

Für das von der FOSBIC Version 2 zugelassene alphanumerische MAT LET-Statement gilt:

$$\underset{234}{\langle \text{Alphanumerisches MAT LET-Statement} \rangle} := \underset{229}{}$$

$$\boxed{\text{MAT LET} \quad \underset{235}{\left\langle \begin{matrix}\text{Alphanumerisches} \\ \text{Zielfeld}\end{matrix} \right\rangle} = \underset{236}{\left\langle \begin{matrix}\text{Alphanumerischer} \\ \text{MAT-Ausdruck}\end{matrix} \right\rangle}}$$

$$\underset{235}{\left\langle \begin{matrix}\text{Alphanumerisches} \\ \text{Zielfeld}\end{matrix} \right\rangle} := \underset{234}{} \langle \text{Feldname} \rangle_{40} \$ \mid \langle \text{Feldname} \rangle_{40} \$$$

$$\langle \text{Stringlängen-Trimmer} \rangle_{223}$$

Vermerkt sei, dass nach ANSI BASIC nur der alphanumerische Feldname zugelassen wird. Es erscheint jedoch wahrscheinlich, dass auch im alphanumerischen Zielfeld der Stringlängen-Trimmer als Norm aufgenommen wird, da er im alphanumerischen MAT-Ausdruck bereits anwendbar ist.[49] Für ihn besteht die Norm

49 ANSI BASIC, p. 52: "Execution of a string-array-assignment shall cause the string-array-expression to be evaluated and its value assigned to the array named to the left of the equals-sign. If appropriate, this array shall be redimensioned dynamically; i. e., its number of dimensions shall be unchanged, but its size shall be changed to conform to the size of the array which is the value of the string-array-expression."

⟨Alphanumerischer MAT-Ausdruck⟩ 236 234 : = (⟨Alphanumerischer Ausdruck⟩) 50 | ⟨Feldname⟩ $ 40 | ⟨Feldname⟩ $ 40

⟨Stringlängen-Trimmer⟩ 223 |

(⟨Alphanumerischer Ausdruck⟩ 250) ⟨Alphanumerischer Operator⟩ 14 ⟨Alphanumerischer MAT-Ausdruck⟩ 236 |

⟨Alphanumerischer MAT-Ausdruck⟩ 236 ⟨Alphanumerischer Operator⟩ 14

((⟨Alphanumerischer Ausdruck⟩) 50 | ⟨Alphanumerische MAT-Funktion⟩ 245

Die Analogie des alphanumerischen MAT-Ausdrucks zum numerischen MAT-Ausdruck (gemäss Gleichung 232) ist unverkennbar, da auch hier die einmalig vorkommende skalare alphanumerische Grösse des alphanumerischen Ausdrucks in Klammern aufgeführt wird und sich dadurch eindeutig von den übrigen Teilen des alphanumerischen MAT-Ausdrucks trennt. Zur Verknüpfung alphanumerischer MAT-Ausdrucke ist nach ANSI BASIC nur das Et-Zeichen (&) als alphanumerischer Operator zulässig.[49a]

Das folgende Beispiel zeigt Textverarbeitungsmöglichkeiten, die sich bei Verwendung der FOSBIC Version 2 ergeben.[50]

```
10 REM MATRIZENTEXTVERARBEITUNG MIT STRINGLAENGENTRIMMER
20 REM FOSBIC VERSION 2
30 DIM A1$17, A2$8, A3$3, A$45, B$(10)5, L$(10)3, X$(10)45
40 A1$="SECHSKANTSCHRAUBE"
50 A2$="DIN 961 "
60 A3$="L="
70 I1=LEN(A1$)
```

49a Die einschlägigen Regelungen von ANSI BASIC (p. 53) lauten wie folgt: "String-arry-expressions [may] involve the [operation] of concatenation. ...Tow string-arrays being concatenated shall have the same number of dimensions and the same size; the concatenation shall be performed element by element. When concatenation is by a scalar, this scalar shall be concatenated with every element of the string-array."

50 Zur Textverarbeitung mit der FOSBIC Version 1 vgl. Weber und Türschmann, BASIC 2, 1. Aufl., S. 81–82.

```
 80 I2=LEN(A2$)
 90 I3=LEN(A3$)
100 A$=A1$
110 A$(I1+6:45)=A2$ & A3$
115 MAT X$=(A$)
120 MAT READ B$(6), L$(6)
130 DATA "M4", "M5", "M6", "M8", "M10", "M12"
140 DATA "30", "40", "50", "60", "70", "80"
150 MAT LET X$(I1+1:I1+5)=B$
160 MAT LET X$=X$ & L$
170 MAT PRINT X$(6)
180 END
```

```
SECHSKANTSCHRAUBE M4  DIN 961 L= 30
SECHSKANTSCHRAUBE M5  DIN 961 L= 40
SECHSKANTSCHRAUBE M6  DIN 961 L= 50
SECHSKANTSCHRAUBE M8  DIN 961 L= 60
SECHSKANTSCHRAUBE M10 DIN 961 L= 70
SECHSKANTSCHRAUBE M12 DIN 961 L= 80
```

Dasselbe Beispiel lässt sich – etwas umständlicher – auch unter Einhaltung der ANSI BASIC-Norm realisieren.
An die Stelle des Statements 140 müsste die Statementfolge

```
140 MAT X$ = (A$(1:I1+1)) & B$
141 MAT X$ = X$ & (A$(I1+6:LEN(A$)))
```

treten.
Bei der Programmkompilation werden vom FOSBIC-Compiler folgende Fehler ausgewiesen:

– Zuweisung alphanumerischer Werte an ein numerisches Feld:

```
        YOU TREAT A NUMERIC VARIABLE AS AN ALPHANUMERIC ONE
SEE BASIC TEXTBOOK  A  323.1
```

– Zuweisung numerischer Werte an ein alphanumerisches Feld

```
        YOU TREAT AN ALPHANUMERIC VARIABLE AS A NUMERIC ONE
SEE BASIC TEXTBOOK  A  323.2
```

331.22 MAT let-Statements

Im Gegensatz zum FOSBIC-Compiler lassen die meisten BASIC-Compiler nur MAT let-Statements zu, wobei wiederum zwischen numerischen und alphanumerischen MAT let-Statements zu unterscheiden ist. Ausserdem wird von ANSI BASIC ein spezielles – nur im Rahmen der Graphikanweisungen[51a] zu verwendendes – graphisches MAT let-Statement vorgesehen.

51 Vgl. Abs. 331.111.

⟨MAT let-Statement⟩ 228 := ⟨Numerisches MAT let-Statement⟩ 238 | ⟨Alphanumerisches MAT let-Statement⟩ 241 |

⟨Graphisches MAT let-Statement⟩ 246a

331.221 Numerisches MAT let-Statement

Für das numerische MAT let-Statement gilt, falls von der bei einzelnen BASIC-Compilern bestehenden Möglichkeit zu Verwendung des ausgeweiteten Alphabets zur Bildung von Feldnamen[51] abgesehen wird:

⟨Numerisches MAT let-Statement⟩ 238 : = 237

MAT let ⟨Numerisches Zielfeld⟩ 231|231a = ⟨Numerischer MAT-Ausdruck⟩ 232

Über die Verfügbarkeit der numerischen MAT let-Statements orientiert Tabelle 6. Speziell zu vermerken ist, dass der FOSBIC-Compiler auch MAT let-Statements zulässt und dass die in Abschnitt 331.21 angeführten Beispielprogramme ohne weiteres unter Verwendung von MAT let-Statements bearbeitet werden können. Damit brauchen an dieser Stelle für die Verwendung der MAT let-Statementtypen

MAT let ⟨Numerisches Zielfeld⟩ = (⟨Arithmetischer Ausdruck⟩)

MAT let ⟨Numerisches Zielfeld⟩ = ⟨Feldname⟩

MAT let ⟨Numerisches Zielfeld⟩ = (⟨Arithmetischer Ausdruck⟩) ⟨Dyadischer Operator⟩ ⟨Feldname⟩

511a Vgl. Bd. 3, Abs. 334.41.

MAT let ⟨Numerisches Zielfeld⟩ = ⟨Feld-name⟩ ⟨Dyadi-scher Operator⟩ (⟨Arith-metischer Ausdruck⟩)

MAT let ⟨Numerisches Zielfeld⟩ = ⟨Feld-name⟩ ⟨Dyadischer MAT-Operator⟩ ⟨Feld-name⟩

keine weiteren Beispiele mehr angeführt zu werden. Dagegen ist den noch nicht diskutierten Statementtypen

MAT let ⟨Numerisches Zielfeld⟩ = ⟨Arithmetische MAT-Funktion⟩

MAT let ⟨Numerisches Zielfeld⟩ = (⟨Arithmetischer MAT-Ausdruck⟩) ⟨Dyadischer Operator⟩

⟨MAT-Initiali-sierungsfunktion⟩

MAT let ⟨Numerisches Zielfeld⟩ = ⟨MAT-Initiali-sierungsfunktion⟩ ⟨Dyadischer Operator⟩

(⟨Arithmetischer Ausdruck⟩)

besondere Aufmerksamkeit zu schenken.
Die arithmetischen MAT-Funktionen wurden zur erleichterten Programmierung oft verwendeter Spezialmatrizen entwickelt. Sie lassen sich in zwei Gruppen einteilen:

⟨Arithmetische MAT-Funktion⟩ 239 232 : = ⟨MAT-Initiali-sierungs-funktion⟩ 240 | ⟨MAT-Verarbei-tungs-funktion⟩ 242

⟨MAT-Initiali-sierungs-funktion⟩ 240 239 : = ⟨Initialisierungs-funktions-name⟩ 241 ((⟨Dimensionierung⟩) 88 |

⟨Initialisierungs-funktions-name⟩ 241 ((⟨Index-Trimmer⟩) 219

⟨Initialisierungs-funktionsname⟩ : = ZER | CON | IDN
241 240

Die MAT-Initialisierungsfunktionen ZER, CON und IDN dienen zur Bildung von Null-, Einer- resp. Einheitsmatrizen;[52] sie können auch zur Neu- oder Redimensionierung des im MAT let-Statement angegebenen Zielfeldes verwendet werden.[53]
Die meisten BASIC-Compiler verlangen bei der Neu- und Redimensionierung mit ZER, CON oder IDN die Angabe der Zeilen- und Spaltenzahl in Form von vorzeichenlosen Ganzzahlen. Beim FOSBIC-Compiler trifft dies nur bei Neudimensionierung einer Matrix zu. Bei der Redimensionierung können sich die aktuellen Feldgrenzen aus einem arithmetischen Ausdruck ergeben, sofern das Ergebnis dieses arithmetischen Ausdrucks – nach Abschneidung eines eventuellen Dezimalteils – positiv und grösser/gleich Eins ist.[54] Weiterhin darf der Index-Trimmer-Wert die ursprünglich aufgrund des DIM-Statements festgelegte maximale Feldgrösse nicht überschreiten.[55]
Im Gegensatz zum DIM-begrenzten MAT READ-[56] und MAT PRINT-

552 ANSI BASIC, p. 50: "The ZER constant shall generate a numeric-array, all of whose elements are zero. The CON constant shall generate a numeric-array, all of whose elements are one. The IDN constant shall generate an identity matrix, i. e., a square matrix with ones on the diagonal and zeroes elsewhere. If a scalar-multiplier is used with either the IDN or CON constants, then each nonzero element of the resultant matrix shall have the value of the numeric-expression."

553 ANSI BASIC, p. 50: "Numeric-array-constants generate numeric-arrays for assignment to the numeric-array to the left of the equals sign. If no redim is present, the size of the numeric-array generated shall be the same as the size of the numeric-array to which it is to be assigned; if a redim is present, a numeric-array of the dimensions specified shall be generated, and the numeric-array to which it is assigned shall be redimensioned accordingly. If a redim is used with the IDN constant, then it shall produce a square matrix; i. e., it shall contain two subscripts whose values are identical." – Zur historischen Entwicklung vgl. Weber und Türschmann, BASIC 2, 1. Aufl., S. 84.

554 Vgl. auch Abs. 331.112.

555 Nach ANSI BASIC (p. 48, 50) ist für den "redim"-Teil generell ein arithmetischer Ausdruck zugelassen.

566 Vgl. Abs. 331.112.

Statement[57] bestehen die aktuellen Feldgrenzen nach Abarbeitung der MAT-Funktion auch weiterhin und sind für das entsprechende Feld im nachfolgenden Programmablauf massgebend. In dem ausschnittsweise wiedergegebenen Beispiel wird durch die MAT-Statements 300 und 400 die ursprünglich dimensionierte Feldgrösse von A (8 Zeilen und 9 Spalten) zunächst auf 7×7 und später auf 4×5 reduziert. Das MAT READ-Statement nimmt jedoch nur Daten für eine 5×6 Matrix auf, während durch das MAT PRINT-Statement nur eine 3×3 Matrix ausgegeben wird. Die beiden letztgenannten MAT-Statements bleiben ohne Auswirkung auf die Feldgrösse.

```
100 DIM A (8,9), B (4,5), C (4,5)        Feldgrösse A  8 × 9
200 MAT READ A (5,6) ←— bearbeitete
                        Feldgrösse
                        5 × 6
    .
    .
    .
300 MAT A = IDN (7,7)                                  7 × 7
    .
    .
    .
400 MAT A = B+C                                        4 × 5
    .
    .
    .
500 MAT PRINT A (3,3) ←— bearbeitete
                        Feldgrösse
    .                   3 × 3
    .
    .
```

Die folgenden MAT let-Statements bewirken, dass allen Elementen des Zielfeldes der Wert Null zugewiesen wird.

```
10 DIM A(3,4), B(5,6)
20 MAT A = ZER
30 MAT B = ZER(4,5)
40 MAT C = ZER(10)
```

Dabei führen die Statements 30 und 40 zu einer Neu- bzw. Redimensio-

57 Vgl. Abs. 331.312.

nnierung.[58] Vermerkt sei, dass die Generation einer vordimensionierten Nullmatrix auch durch das Statement

```
20 MAT A = (0)
```

eerfolgen kann.

Zur Bildung einer Einermatrix dienen die folgenden MAT let-Statements,

```
10 DIM D(5,5), E(15)
20 MAT D = CON
30 MAT E = CON (10)
40 MAT F = CON(3,3)
```

wobei Statement 40 auch die Neudimensionierten der Matrix F bewirkt. Anstelle von Statement 20 kann

```
20 MAT D = (1)
```

ggesetzt werden, wie auch das Beispielprogramm auf S. 79 zeigt. Die Bildung einer Einheitsmatrix erfolgt durch die Statements

```
30 MAT G = IDN
40 MAT H = IDN(5,5)
```

Dabei muss die Matrix G bereits als quadratische Matrix dimensioniert sein. Zur Matrizenredimensionierung vgl. auch das folgende Beispielprogramm.

```
10 REM REDIMENSIONIERTE EINHEITSMATRIX
20 DIM I(5,5)
30 READ N
40 MAT I = IDN(SQR(N), SQR(N))
50 MAT PRINT I
60 DATA 10
70 END

MATRIX *I* ZEILE *   1*
    1              0.0              0.0
MATRIX *I* ZEILE *   2*
    0.0            1                0.0
MATRIX *I* ZEILE *   3*
    0.0            0.0              1
```

58 NCC Standard BASIC, p. 132: "When an array destination is dynamically redimensioned, the bounds on the first and second subscripts need not be individually less than the bounds specified in the dimension statement for that array as long as the total space required for the redimensioned array is at most that reserved for the array be the dimension statement. Dynamic redimensioning may not change the number of dimensions of an array."

Für die MAT-Verarbeitungsfunktionen gilt:

$$\langle \text{MAT-Verarbeitungsfunktion} \rangle_{242,\,239} := \langle \text{Verarbeitungsfunktionsname} \rangle_{243} \; (\langle \text{Feldname} \rangle_{40}) \mid$$

$$\langle \text{Verarbeitungsfunktionsname} \rangle_{243} \; (\langle \text{Feldname} \rangle_{40} \, , \, \langle \text{Feldname} \rangle_{40}) \mid$$

$$\langle \text{Verarbeitungsfunktionsname} \rangle_{243}$$

$$\langle \text{Verarbeitungsfunktionsname} \rangle_{243,\,242} := \text{TRN} \mid \text{INV} \mid \text{DET} \mid \text{DOT} \mid \S\S\S\S$$

Die explizit angeführten Verarbeitungsfunktionsnamen sind nach ANSI BASIC vorgesehen.[58a] Durch §§§§ seien Namen von MAT-Verarbeitungsfunktionen dargestellt, die bei einzelnen BASIC-Compilern zusätzlich verfügbar sind.[59]
Die MAT-Verarbeitungsfunktionen TRN und INV dienen zur Matrizentransposition bzw. -Inversion, wobei die zu bearbeitende Matrix mit ihrem Namen bezeichnet sein muss. Vgl. hierzu die folgenden Beispiele.

58a ANSI BASIC, p. 50: "The function TRN shall produce the transpose of its argument. The function INV shall produce the inverse of its argument, which shall be a square numeric matrix. When used without an argument, the numeric-function DET shall return the value of the determinant of the numeric matrix most recently used as an argument of an INV function. When used with an argument, DET shall return the value of the determinant of this argument, which shall be a square numeric matrix. ...The value of DOT (X,Y) shall be the inner product of X and Y, which shall be the names of single-subscripted numeric-arrays."

59 Vgl. Bennet P. Lientz. A Comparative Evaluation of Versions of BASIC. Communications of the ACM 19 (1976) 4, April, p. 176–177; PRD (product of elements in an array), SUM (sum of elements in an array), ASORT (matrix with elements in ascending order), DSORT (matrix with elements in descending order). – IBM. IBM System/370. VS BASIC Language. 2nd ed., New York 1974, p. 90–91, 107–109.

```
10 REM MATRIZENTRANSPOSITION
20 DIM A(2,3), B(3,2)
30 MAT INPUT A
40 MAT B = TRN(A)
50 MAT PRINT A
70 MAT PRINT B
80 END LIST DATA

LIST OF DATA AFTER END STATEMENT

1,2,3,10,20,30
*

MATRIX *A* ZEILE *   1*
   1              2              3
MATRIX *A* ZEILE *   2*
  10             20             30
MATRIX *B* ZEILE *   1*
   1             10
MATRIX *B* ZEILE *   2*
   2             20
MATRIX *B* ZEILE *   3*
   3             30
```

```
10 REM MATRIZENINVERSION
20 DIM A(2,2), B(2,2)
30 MAT READ A
40 MAT PRINT A
50 MAT B = INV(A)
60 MAT PRINT B
70 DATA 1,4,2,1
80 END

 1              4
 2              1
-.142857        0.571429
0.285714        -.142857
```

Mit dem vorstehenden Inversionsprogramm wird

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 1 \end{bmatrix} \text{ eingelesen und invertiert, so dass}$$

$$B = \begin{bmatrix} -1/7 & 4/7 \\ 2/7 & -1/7 \end{bmatrix}$$

Vermerkt sei, dass grundsätzlich auch die Möglichkeit besteht, die Werte der Kehrmatrix der entsprechenden Ursprungsmatrix zuzuweisen, was allerdings bei den meisten BASIC-Prozessoren nicht erlaubt ist.[59a]

59a Control Data, BASIC Version 3. Reference Manual, p. 7.3; Honeywell Bull. Time-Sharing Service MARK III. BASIC Language. Köln 1972, S. 156–157; Nixdorf, System- Software Business- Basic, S. 10.25–27. – Interessant ist bei Nixdorf die Möglichkeit, mit den MAT-Funktionen INV und TRN gleichzeitig auch neu zu dimensionieren, da bis zur Ausführung dieser MAT-Funktionen die Feldgrenzen der Matrix auf der linken Seite (numerisches Zielfeld) nicht festgelegt zu sein brauchen.

```
10 DIM A(2,2)
20 MAT INPUT A
30 MAT A = INV(A)
40 MAT PRINT A
50 END LIST DATA

LIST OF DATA AFTER END STATEMENT

1,4,2,1
```

Aufgrund dieses Programms ergibt sich:

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} -0.1428 & 0.5714 \\ 0.2857 & -0.1428 \end{bmatrix}$$

Der Versuch zur Inversion einer singulären Matrix führt in der Regel zum Programmabbruch mit entsprechender Fehlermeldung.
Durch die MAT Funktion INV können insbesondere auch lineare Gleichungssysteme komfortabel bearbeitet werden, wie das folgende Beispiel zeigt.

```
10 REM MATRIZENINVERSION
20 REM LOESUNG EINES LINEAREN GLEICHUNGSSYSTEMS
30 DIM A(4,4), B(4,1), C(4,4), X(4,1)
40 MAT READ A, B
50 MAT C = INV(A)
60 MAT X = C*B
70 PRINT ≠LOESUNGSVEKTOR X≠
80 PRINT ≠----------------≠
90 MAT PRINT X
92 DATA 2,3,-1,0,-6,-5,0,2,2,-5,6,-6,4,6,2,-3
94 DATA 20,-45,-3,58
120 END

LOESUNGSVEKTOR X
----------------
        1
        7
        3
       -2
```

Mit dem FOSBIC-Compiler sind die arithmetischen MAT-Funktionen ZER, CON, IDN, TRN und INV in allen Versionen verarbeitbar, wobei mit der MAT-Funktion INV Matrizen bis maximal 12 × 12 Elementen invertiert werden können. Ausserdem wird bei den MAT-Verarbeitungsfunktionen TRN und INV zugelassen, dass Zielfeld und Herkunftsfeld den gleichen Feldnamen tragen.
Bei einzelnen BASIC-Compilern besteht auch die Möglichkeit, die Determinante DET der zuletzt invertierten Matrix ausdrucken zu lassen. Nach ANSI BASIC kann die Funktion DET mit oder ohne Argument verwendet werden. Ohne Argument liefert DET den Determinantenwert der zuletzt invertierten Funktion, mit Argument den Determinantenwert

des Argumentes. Eine singuläre Matrix oder nicht invertierbare Matrix soll in DET den Wert Null liefern.[60]

Von besonderer Bedeutung ist die MAT-Funktion DOT, die immer mit zwei Argumenten aufgeführt werden muss, wobei die beiden Feldnamen einfach indizierte Variable bezeichnen. DOT bildet das innere oder skalare Produkt der beiden im Argument aufgeführten Vektoren. Mit dieser MAT-Funktion umgeht ANSI BASIC das Problem, auf der rechten Seite einer MAT-Wertzuordnungsanweisung einfache numerische Variable zuzulassen, die bei einem Skalarprodukt zwangsläufig entstehen.[61]

Die nachstehenden Programmabschnitte zeigen mehrere Möglichkeiten zur Bildung des Skalarproduktes zweiter Vektoren auf, wobei nach ANSI BASIC alle Formen ausser der letzten zulässig sind.

```
100 DIM X(9), Y(9)
110 A1=DOT(X,Y)
```

ODER

```
100 DIM A(1,1), X(1,9), Y(9,1)
110 MAT A = X * Y
```

ODER

```
100 DIM A(1,1), X(9), Y(9,1)
110 MAT A = X * Y
```

ODER

```
100 DIM X(9), Y(9,1)
100 MAT A1 = X * Y
```

Die obigen Beispiele zur Bildung des inneren Produktes zeigen, dass nur die erste Form aus der in der Matrizenalgebra üblichen Form herausfällt. Es bleibt daher zu hoffen, dass die zukünftige BASIC-Entwicklung sich wieder an der Matrizenalgebra orientiert und die Entwicklung von Übergangsfunktionen zwischen Vektoren und Skalaren unterbleibt.

Der FOSBIC-Compiler orientiert sich an den Grundlagen der Matrixalgebra.

Wermerkt sei, dass nach den Vorstellungen der ANSI X3J2 Matrix Subkommission die Leistungsfähigkeit der BASIC-Compiler dahingehend

60 ANSI BASIC, p. 50: "An implementation shall not terminate execution of the program if the argument of the INV function is a singular matrix. Instead, the value of DET after such an inversion shall be zero, and the value of each element of the numeric-array to the left of the equals sign sall be implementation-defined."

61 Vgl. dazu auch die Beispiele auf S. 85.

ausgeweitet werden sollte, dass auch kombinierte MAT-Operationen möglich sind.
Beispielsweise sollten die Statements

```
110 MAT A = TRN(B)
120 MAT D = A*C
130 MAT A = INV(D)
```

durch das Statement

```
110 MAT A = INV(TRN(B)*C)
```

ersetzt werden können.[62]
Wie bereits erwähnt, werden vom FOSBIC-Compiler via MAT let- (und MAT LET-)Statements auch MAT-Funktionen vom Typ ZER, CON, IDN, TRN und INV verarbeitet. Bei der Programmausführung werden folgende Fehler nachgewiesen:

– Dimensionierungsfehler bei IDN:

```
PROGRAM STOPPED -- MATRIX *A* IS NOT DIMENSIONED N*N ROWS=    24 COLUMNS=    25
                   ERROR CODE= 35
                ***** SEE BASIC TEXTBOOK =  B331.211  *****
                ***** I WAS AT LINE NUMBER    0
```

– Dimensionierungsfehler bei TRN:

```
PROGRAM STOPPED -- NO CORRESPONDING SIZE OF THE SUBSCRIPTS FOR MATRIX TRANSPONATION
                   ERROR CODE= 31
                ***** SEE BASIC TEXTBOOK =  B331.211  *****
                ***** I WAS AT LINE NUMBER    0
```

– Dimensionierungsfehler beim Zielfeld quadratischer Matrizen:

```
PROGRAM STOPPED -- ILLEGAL REDIMENSION OF THE MATRIX *A* ROWS=
                   24 COLUMNS=    25  MAXIMAL STORAGE ELEMENTS    345.
                   ERROR CODE= 34
                ***** SEE BASIC TEXTBOOK =  B331.312  *****
                ***** I WAS AT LINE NUMBER    0
```

– Fehler in korrespondierenden Feldgrenzen:

```
PROGRAM STOPPED -- SIZE ERROR IN SUBSCRIPTS OF THE VARIABLES *  ROWS   * COLUMNS
                                                          A       3         3
                                                          B       4         5
                                                          C       2         3

PROGRAM STOPPED -- ERROR IN MATRIX OPERATION -- ADDITION OR SUBTRACTION
                   ERROR CODE= 36
                ***** SEE BASIC TEXTBOOK =  B331.221  *****
                ***** I WAS AT LINE NUMBER    0
```

62 X3J2/75-23, G-10. – Nach ANSI BASIC ist diese Form der Matrizengleichungen im Rahmen der MAT let-Statements noch nicht erlaubt. Bei den Graphikanweisungen besteht jedoch die Möglichkeit, sogenannte Transformationsgleichungen als MAT let-Statements anzugeben, die in einfacher Form bereits kombinierte MAT let-Statements darstellen ; vgl. Abs. 331.223.

-- Dimensionierungsfehler bei INV:

```
PROGRAM STOPPED -- NO CORRESPONDING SIZE OF THE SUBSCRIPTS FOR MATRIX INVERSION
                   ERROR CODE= 48
                ***** SEE BASIC TEXTBOOK =  8331.221  *****
                ***** I WAS AT LINE NUMBER    0
```

-- Determinante der zu invertierenden Matrix ist Null:

```
PROGRAM STOPPED -- DETERMINANT OF THE MATRIX *X* IS ZERO
                   ERROR CODE= 49
                ***** SEE BASIC TEXTBOOK =  8331.221  *****
                ***** I WAS AT LINE NUMBER    0
```

-- Zu invertierende Matrix ist zu gross:

```
PROGRAM STOPPED -- I CAN ONLY INVERT A  24 *  24 MATRIX
                   ERROR CODE= 50
                ***** SEE BASIC TEXTBOOK =  8331.221  *****
                ***** I WAS AT LINE NUMBER    0
```

331.222 Alphanumerisches MAT let-Statement

Für alphanumerische MAT let-Statements gilt generell:

⟨Alphanumerisches MAT let-Statement⟩ : =
244 237

MAT let ⟨Alphanumerisches Zielfeld⟩ 235 = ⟨Alphanumerischer MAT-Ausdruck⟩ 236

Diese Strukturgleichung ist bis auf das fehlende Kennwort LET identisch mit Gleichung 234 und entspricht in dieser Form der ANSI BASIC-Norm.[62a]

Bezüglich der – in Abschnitt 331.212 noch nicht behandelten – alphanumerischen MAT-Funktion gilt:

⟨Alphanumerische MAT-Funktion⟩ : = ⟨Funktions-name⟩ $ |
245 236 246

⟨Funktions-name⟩ $ (⟨Dimensio-nierung⟩) |
246 88

62a ANSI BASIC, p. 52–53.

⟨Funktions-name⟩ $ (⟨Index-Trimmer⟩) |
246 219

⟨Funktions-name⟩ $ (⟨Dimensio-nierung⟩) ⟨Maximale Stringlänge⟩ |
246 88 93

⟨Funktions-name⟩ $ ⟨Maximale Stringlänge⟩ (⟨Dimensio-nierung⟩) |
246 93 88

⟨Funktions-name⟩ $ ⟨Stringlängen-Trimmer⟩ |
246 223

⟨Funktions-name⟩ $ (⟨Index-Trimmer⟩) ⟨Stringlängen-Trimmer⟩ |
246 219 223

⟨Funktions-name⟩ $ ⟨Stringlängen-Trimmer⟩ (⟨Index-Trimmer⟩ |
246 223 219

Nach ANSI BASIC sind nur die drei ersterwähnten Formen genormt. Als Funktionsname wird NUL verwendet; falls eventuell compilerspezi fisch verfügbare weitere MAT-Funktionsnamen mit §§§§ bezeichne werden, gilt:

⟨Funktionsname⟩ : = NUL | §§§§
246 245

Durch die NUL-Funktion wird ein String mit Leerzeichen generiert.[63] Bei der FOSBIC Version 2 ist die NUL-Funktion verfügbar, wobei die aus Gleichung 245 ersichtlichen Formen der alphanumerischen MAT-Funktion vollumfänglich realisierbar sind.[63a]

3311.223 *Graphisches MAT let-Statement*

Das graphische MAT let-Statement kann nur in Verbindung mit anderen Graphikanweisungen und dann nur mit Matrizen der Dimension 4 × 4 verwendet werden; es wird im ANSI BASIC-Bericht auch nur in diesem Zusammenhang erwähnt.[63b]

Diese Struktur des graphischen MAT let-Statements unterliegt folgender Vorschrift:[63c]

⟨Graphisches MAT let-Statement⟩ : =
246a 237

MAT ⟨Feldname⟩ = ⟨Transformationsausdruck⟩
40 368

Der Transformationsausdruck besteht aus einem oder mehreren, durch Sternzeichen (*) miteinander verbundenen, Feldnamen oder Transformationsfunktionen.[63d] Das graphische MAT let-Statement kann in dieser Form bereits als kombiniertes MAT let-Statement aufgefasst werden.[63e]

663 ANSI BASIC, p. 53: "The string-array-constant NUL shall generate an array all of whose elements are the null string. If no redim is present, this array shall have the same size as that of the string-array to which it is to be assigned. If a redim is present, this array shall have the size specified by the redim." – Nach ANSI BASIC 82 (p. 56–57) wird NULL als Funktionsname verwendet.

663a Bezüglich der FOSBIC Version 1 vgl. Weber und Türschmann, BASIC 2, 1. Aufl., S. 90.

663b ANSI BASIC, p. 115; das Statement wird dort als "transform-assignment" bezeichnet.

663c ANSI BASIC, p. 115: "transform-assignment = MAT numeric-array-name equal-sign transform."

663d Vgl. Abs. 334.41.

663e Vgl. S. 101–102.

331.3 MAT-Ausgabeanweisungen

Die MAT-Ausgabeanweisungen können als MAT PRINT- oder MAT PRINT USING-Statements realisiert werden. Vgl. Tabelle 7. Sie bewirken – ohne Spezifikation der einzeln auszugebenden Matrizenelemente – die Übertragung ganzer Felder auf vorgegebene Datenträger.[64]

⟨MAT-Ausgabe-anweisung⟩ 248 208 := ⟨MAT PRINT-Statement⟩ 249 | ⟨MAT PRINT USING-Statement⟩ 257

331.31 MAT PRINT-Statements

MAT PRINT-Statements können sowohl zur Druckausgabe als auch zur Sichtanzeige auf Terminalbildschirmen verwendet werden. Die Elemente der auszugebenden Matrix werden zeilenweise auf den Datenträger übernommen, wobei jede Matrizenzeile als Einheit behandelt wird.[65]
Zur Steuerung der MAT-Ausgabe dienen die MAT-Ausgabelisten. Sie enthalten die – normalerweise durch Kommata voneinander abgetrennten – Namen der auszugebenden Felder.
Numerische und alphanumerische Felder werden in der Reihenfolge ihres Auftretens in der MAT-Ausgabeliste verarbeitet. Normalerweise wird die Standard-Druckzoneneinteilung – fünf Zonen zu je fünfzehn Zeichen[66] – bei der Datenausgabe eingehalten; in Analogie zur elementaren

64 Vgl. auch Bd. 1, Abs. 332.21 und Abs. 332.23.

65 NCC Standard BASIC, p. 135: "The mat-print statement causes entire arrays (except for rows and columns zero) to be printed row by row." – ANSI BASIC, p. 99: "Execution of an array-print-statement shall cause the values of all elements in all arrays in the array-print-list to be printed."

66 NCC Standard BASIC, p. 135: "[Normally], the elements of each row are printed in columns 15 spaces wide. Each row begins on a new line. If a row does not fit on line, it is continued on succeeding lines." – ANSI BASIC, p. 99: "The characters generated for transmission to an external device by the printing of a single array in an array-print-list are almost precisely those that would be generated if the elements in that array had been listed, row by row, in the print-list or formatted-print-list of print-statement, separated by the separator which follows the array-name in the array-print-list (or separated by a comma if no separator follows the array-name). The only additional characters generated shall be an end-of-line each time a row of the array has been printed (if such an end-of-line has not already been generated)." – Zur gepackten Datenausgabe vgl. Bd. 1, Abs. 332.21.

Druckausgabe kann durch den Strichpunkt (;) als Trennungszeichen eine gepackte Datenausgabe programmiert werden.[67]
Vektoren werden, ihrer Struktur entsprechend, zeilenweise ausgegeben.[68] Sich folgende Matrizen werden durch eine Leerzeile auseinandergehalten.[69]
Im übrigen bieten einzelne BASIC-Compiler auch die Möglichkeit zur MAT-Redimensionierung via MAT-Ausgabeliste. Es kann damit – ähnlich wie bei den MAT-Eingabeanweisungen – zwischen DIM-gebundenen und DIM-begrenzten MAT PRINT-Statements unterschieden werden.

⟨MAT PRINT-Statement⟩ 249 248 : = ⟨Allgemeines MAT PRINT-Statement⟩ 250 |

⟨DIM-gebundenes MAT PRINT-Statement⟩ 253 | ⟨DIM-begrenztes MAT PRINT-Statement⟩ 255

Für das allgemeine MAT PRINT-Statement gilt:

⟨Allgemeines MAT PRINT-Statement⟩ 250 : = 249

MAT PRINT ⟨MAT-Ausgabeliste⟩ 251

⟨MAT-Ausgabeliste⟩ 251 250 : = ⟨DIM-gebundene MAT-Ausgabeliste⟩ 254 | ⟨DIM-gebundene MAT-Ausgabeliste⟩ 254 |

67 NCC Standard BASIC, p. 135: "If an array is followed by a semicolon separator, the elements are closely packed."
68 Andere Regelung nach NCC Standard BASIC, p. 135: "If no print separator follows a vector, it is printed as a column vector, i. e. one element per line; otherwise it is printed as a row vector."
69 NCC Standard BASIC, p. 135: "A blank line is printed between successive arrays in a mat-print list." – ANSI BASIC, p. 99: "An extra end-of-line shall be generated between the output for successive arrays in an array-print-list."

⟨MAT-Ausgabe-liste⟩ 251 , ⟨MAT-Ausgabe-liste⟩ 251 |

⟨MAT-Ausgabe-liste⟩ 251 ; ⟨MAT-Ausgabe-liste⟩ 251 |

⟨MAT-Ausgabe-liste⟩ 251 , ⟨MAT-Schluss-liste⟩ 252 |

⟨MAT-Ausgabe-liste⟩ 251 ; ⟨MAT-Schluss-liste⟩ 252 |

⟨MAT-Schlussliste⟩ 252

⟨MAT-Schlussliste⟩ 252 : = ⟨MAT-Ausgabeliste⟩ 251 |

⟨MAT-Ausgabeliste⟩ 251 , |

⟨MAT-Ausgabeliste⟩ 251 ;

Nach ANSI BASIC ist nur das DIM-gebundene MAT PRINT-Statement erlaubt.[70] Grundsätzlich unmöglich ist die Neudimensionierung einer Matrix im MAT PRINT-Statement.

70 ANSI BASIC, p. 97.

Prozessor	ungepackt	gepackt	numerisch	MAT PRINT alpha-numerisch	gemischt	Redimen-sionierung/ Art	MAT PRINT USING
ANSI BASIC 79	×	×	×	×	×	nein	ja
Dartmouth BASIC-6	×	×	×			nein	ja
UOM BASIC UBASIC	×	×	×	×	×	nein	nein
IBM CALL BASIC	×	×	×			nein	nein
FOSBIC	×	×	×	×	×	arithmet. Ausdruck	nein
Wang Laboratories	×	×	×	×	×	Ganzzahl	nein
Nixdorf	×	×	×	×		nein	nein
Digital Equipment	×	×	×	×	×	nein	nein

Tabell 7. Verfügbarkeit der MAT-Wertzuordnungsanweisungen

331.311 DIM-gebundenes MAT PRINT-Statement

Das DIM-gebundene MAT PRINT-Statement weist die Form

⟨DIM-gebundenes MAT PRINT-Statement⟩ : =
253 249

MAT PRINT ⟨DIM-gebundene MAT-Ausgabeliste⟩ 254

auf, wobei gilt

⟨DIM-gebundene MAT-Ausgabeliste⟩ : = ⟨Feldname⟩ | ⟨Feldname⟩ $|
254 251|253 40 40

⟨DIM-gebundene MAT-Ausgabeliste⟩ , ⟨MAT-Ausgabeliste⟩ |
254 251

⟨DIM-gebundene MAT-Ausgabeliste⟩ ; ⟨MAT-Ausgabeliste⟩ |
254 251

Aufgrund des nachfolgenden Beispielprogramms wird die Matrix A in normaler Druckdarstellung (ungepackt) ausgegeben. Das wiedergegebene Druckbild ist FOSBIC-spezifisch und dadurch charakterisiert, dass jede Matrizenzeile automatisch gekennzeichnet werden kann.

```
10 REM FOSBIC-SPEZIFISCHER MAT DRUCK
20 DIM A(3,4)
30 MAT READ A
40 MAT PRINT A
50 DATA 1,2,3,4,5,6,7,8,9,10,11,12
60 END

MATRIX "A" ZEILE "   1"
    1          2          3          4
MATRIX "A" ZEILE "   2"
    5          6          7          8
MATRIX "A" ZEILE "   3"
    9         10         11         12
```

Wird in der dem BASIC-Programm vorausgehenden Stern-Karte ab der dritten Spalte der Text HEADLINE angegeben, so erfolgt die Kennzeichnung der einzelnen Matrizenzeilen. Vgl. hierzu das folgende Beispiel, bei dem auch von der Möglichkeit zur gepackten Matrizendarstellung Gebrauch gemacht wird.

```
* NO-HEADLINE
10 REM GEPACKTER MAT-DRUCK
20 DIM A(5,5)
30 M = 4
40 N = 4
50 MAT READ A(M,N)
60 MAT PRINT A(M,N):
70 DATA 1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,
80 END
*

1 2 3 4
5 6 7 8
9 10 11 12
1 2 3 4
```

Der FOSBIC-Compiler lässt die gepackte Ausgabe für numerische und alphanumerische Felder zu, wobei die Reihenfolge der Felder beliebig gewählt werden kann.

```
10 REM TEXTAUSGABE
20 REM FOSBIC-SPEZIFISCHE STRINGLAENGE BEACHTEN
30 MAT READ A$(5)
35 MAT READ B(5)
40 MAT READ C$(5)
45 MAT READ D(5)
50 MAT PRINT A$
55 MAT PRINT B
60 MAT PRINT C$: D:
70 DATA #UNGEPACKTE TEXTAUSGABE#
75 DATA 5,4,3,2,1
80 DATA #GEPACKTE TEXTAUSGABE  #
85 DATA 1,2,3,4,5
90 END

    VEKTOR *A*
UNGEP          ACKTE          TEXT          AUSGA          BE
    VEKTOR *B*
     5              4              3              2              1
    VEKTOR *C*
GEPACKTE TEXTAUSGABE
    VEKTOR *D*
1 2 3 4 5
```

Vom FOSBIC-Compiler werden bei der Programmkompilierung folgende Fehler erkannt:

- Kennwortfehler bei MAT:

```
        YOU MAY UNDERSTAND THE COMMAND ABOVE BUT I DO NOT
SEE BASIC TEXTBOOK  A  326.1
```

- Kennwortfehler bei PRINT:

```
        I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B    331
```

- Fehlende Dimensionierung:

```
      - VARIABLE *D* IS NOT DIMENSIONED
        I CANNOT ACCEPT THE MAT STATEMENT ABOVE
SEE BASIC TEXTBOOK  B331.111
```

- Unzulässige Angabe von Feldgrenzen:

```
        DIMENSION NOT ALLOWED WITH EXPRESSION
        I DO NOT UNDERSTAND THE FOLLOWING PART IN THE MAT STATEMENT ABOVE

SEE BASIC TEXTBOOK  B331.211
```

331.312 DIM-begrenztes MAT PRINT-Statement

Das DIM-begrenzte MAT PRINT-Statement ist bei einzelnen BASIC-Compilern in der Form

⟨DIM-begrenztes MAT PRINT-Statement⟩ : =
255 249

MAT PRINT ⟨DIM-begrenzte MAT-Ausgabeliste⟩ 256

zulässig, wobei gilt:

⟨DIM-begrenzte MAT-Ausgabeliste⟩ : = ⟨Feldname⟩ (⟨Index-Trimmer⟩) | ⟨Feldname⟩ $ (⟨Index-Trimmer⟩)
256 251|255 40 219 40 219

⟨DIM-begrenzte MAT-Ausgabeliste⟩ , ⟨MAT-Ausgabeliste⟩ |
256 251

⟨DIM-begrenzte MAT-Ausgabeliste⟩ ; ⟨MAT-Ausgabeliste⟩ |
256 251

Nach NCC Standard BASIC bezieht sich die Redimensionierung auf die mit dem MAT PRINT-Statement auszugebende Matrix.[71] Beim FOSBIC-Compiler erfolgt ebenfalls die Ausgabe des Feldes nach den angegebenen Feldgrenzen, während die Redimensionierung für die bearbeitete Matrix in den nächstfolgenden MAT-Statements nicht wirksam bleibt. Vgl. hierzu das folgende Beispiel.

```
10 MAT READ A(3,6)
20 MAT PRINT A(2,5)
40 MAT PRINT A
50 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
60 END
```

71 NCC Standard BASIC, p. 135.

```
MATRIX *A* ZEILE *   1*
   1              2              3              4              5
MATRIX *A* ZEILE *   2*
   7              8              9             10             11
MATRIX *A* ZEILE *   1*
   1              2              3              4              5
   6
MATRIX *A* ZEILE *   2*
   7              8              9             10             11
  12
MATRIX *A* ZEILE *   3*
  13             14             15             16             17
  18
```

Eine bleibende, für alle nachfolgenden MAT-Statements geltende Redimensionierung der entsprechenden Matrix ist nur durch MAT-Funktionen innerhalb einer MAT-Wertzuordnungsanweisung möglich.[72] Eine Redimensionierung durch MAT-Funktionen ist beliebig oft möglich. Um für die Matrix A im obigen Beispiel eine fortdauernde Redimensionierung zu erzielen, muss demnach folgende Statementfolge angegeben werden:

```
10 MAT READ A(3,6)
20 MAT PRINT A(2,6)
25 MAT A = ZER(4,4)
30 RESTORE
35 MAT READ A
40 MAT PRINT A
50 DATA 1,2,3,4,5,6,7,8,9,10;11,12,13,14,15,16,17,18
60 END
MATRIX *A* ZEILE *   1*
   1              2              3              4              5
   6
MATRIX *A* ZEILE *   2*
   7              8              9             10             11
  12
MATRIX *A* ZEILE *   1*
   1              2              3              4
MATRIX *A* ZEILE *   2*
   5              6              7              8
MATRIX *A* ZEILE *   3*
   9             10             11             12
MATRIX *A* ZEILE *   4*
  13             14             15             16
```

Beachtenswert ist auch, dass zwar alle MAT PRINT-Statements eine vorausgehende Dimensionierung verlangen, dass diese Dimensionierung aber nicht unbedingt durch ein DIM-Statement erfolgen muss. Beispiel:

```
10 REM FOSBIC-KONFORME NEU-/REDIMENSIONIERUNG
20 MAT READ A$(6)
30 MAT PRINT A$(1)
40 MAT PRINT A$(3)
50 MAT PRINT A$;
60 DATA #STRINGLAENGE FUENF ZEICHEN#
70 END

    VEKTOR *A*
STRIN
    VEKTOR *A*
STRIN          GLAEN          GE FU
    VEKTOR *A*
STRINGLAENGE FUENF ZEICHEN
```

72 Vgl. 331.221.

Bei Verarbeitung DIM-begrenzter MAT PRINT-Statements werden vom FOSBIC-Compiler – ausser den im vorigen Abschnitt erwähnten Fehlern – bei der Programmausführung folgende Unstimmigkeiten moniert:[73]

– Unrichtige Redimensionierung, Zeilenzahl:

```
PROGRAM STOPPED -- ROW INDEX OF THE MATRIX *A* IS NEGATIVE OR ZERO   -12345.00
                   ERROR CODE= 33
              ***** SEE BASIC TEXTBOOK =  B331.312  *****
              ***** I WAS AT LINE NUMBER    0
```

– Unrichtige Redimensionierung, Spaltenzahl:

```
PROGRAM STOPPED -- COLUMN INDEX OF THE MATRIX *A* IS NEGATIVE OR ZERO   -12345.00
                   ERROR CODE= 32
              ***** SEE BASIC TEXTBOOK =  B331.312  *****
              ***** I WAS AT LINE NUMBER    0
```

– Unrichtige Redimensionierung, zu hoher DIM-Wert:

```
PROGRAM STOPPED -- ILLEGAL REDIMENSION OF THE MATRIX *A* ROWS=
                   24 COLUMNS=   25  MAXIMAL STORAGE ELEMENTS   345.
                   ERROR CODE= 34
              ***** SEE BASIC TEXTBOOK =  B331.312  *****
              ***** I WAS AT LINE NUMBER   10
```

331.32 MAT PRINT USING-Statements

Analog zu den elementaren PRINT-Anweisungen verfügen die MAT-Ausgabeanweisungen auch über ein formatiertes MAT PRINT-Statement, welches hier konsequenterweise als MAT PRINT USING-Statement bezeichnet wird.[74] Es unterliegt der ANSI BASIC-Norm[75], wobei allerdings in der MAT-Ausgabeliste nur die DIM-gebundene Form zugelassen wird.

⟨MAT PRINT USING-Statement⟩ : =
257 248

MAT PRINT USING ⟨Image-Statement-kennung⟩ 131 : ⟨MAT-Ausgabe-liste⟩ 251 |
MAT PRINT USING ⟨Image-Statement-kennung⟩ 131 , ⟨MAT-Ausgabe-liste⟩ 251

Während bei der normalen PRINT- und MAT PRINT-Druckausgabe durch das Semikolon (;) in der Ausgabeliste die gepackte Druckausgabe erzeugt wird[77], unterbindet es im MAT PRINT USING-Statement den Zeilenvorschub zwischen zwei Matrizenausgaben; die Ausgabe der nächsten Matrix und/oder des nächsten Druckelements erfolgt somit unmittelbar hinter der letzten Druckausgabe.[78]

```
10 DIM A(4,4)
20 MAT READ A
30 MAT PRINT USING 50: A
40 PRINT "MATRIX A"
50 ! ## ## ## ##
60 DATA 1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4
70 END
```

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
 1  2  3  4
MATRIX A
```

Endet das MAT PRINT USING-Statement mit einem Semikolon, so ergibt sich für das obige Beispiel mit dem Statement

```
30 MAT PRINT USING 50: A;
```

das folgende Druckbild

```
1  2  3  4
5  6  7  8
9 10 11 12
1  2  3  4 MATRIX A
```

Der FOSBIC-Compiler lässt das MAT PRINT USING-Statement erst ab Version 2 zu.

73 Vgl. Abs. 331.112.
74 Vgl. Bd. 1, Abs. 332.23.
75 ANSI BASIC, p. 97.
76 ANSI BASIC, p. 97, 92.
77 Vgl. Bd. 1, Abs. 332.23.
78 ANSI BASIC, p. 95: "An end-of-line shall be generated after all other character generation is completed, unless the output-list ends with a semicolon, in which case no such end-of-line shall be generated."

332 Fileanweisungen

Im Rahmen der Programmiersprache BASIC ist generell zwischen systeminternen und benutzereigenen Dateien zu unterscheiden.
Systeminterne Dateien werden bei der Bearbeitung von BASIC-Programmen via DATA-Statements von den BASIC-Prozessoren automatisch angelegt und mit Hilfe entsprechender Eingabeanweisungen (READ- und MAT READ-Statements) benutzt. Diese Dateien sind in ihrer Struktur für den BASIC-Benutzer fest vorgegeben. Sie werden als Basisdateien bezeichnet.
Im allgemeinen wird pro BASIC-Programm nur eine Basisdatei angelegt;[79] sie befindet sich im Hauptspeicher und ist sequentiell organisiert. Verwiesen sei auf die diesbezüglichen Ausführungen in Band 1.[80]
Neben den vorerwähnten systeminternen Dateien lassen die meisten BASIC-Prozessoren auch die Verwendung benutzereigender Dateien zu. Diese werden physikalisch auf peripheren Datenträgern realisiert und sind logisch vom BASIC-Benutzer mit Hilfe von Fileanweisungen (FL-Anweisungen) zu strukturieren und zu verwalten.
Die folgenden Ausführungen beziehen sich ausschliesslich auf das logische Zusammenwirken von Daten und BASIC-Programmen im Rahmen benutzereigener Dateien (files). Die physikalische Realisation von Dateien auf peripheren Datenträgern (Magnetbändern, Magnetplatten, etc.) erfolgt durch das Betriebssystem und wird nicht in die Diskussion einbezogen. Vgl. hierzu auch den entsprechenden Ansatz von ANSI BASIC.[81]

79 Vgl. auch ANSI X3J2 Files Subcommittee (X3J2/75-23, G-5): "[Basic files are] associated with the 'standard' input and output streams of the BASIC program... [and they are] adressed by Minimal BASIC PRINT- and INPUT-Statements."

80 Vgl. Bd. 1, Abs. 332.11.

81 ANSI BASIC, p. 54: "Files are collections of data external to BASIC programs. Particular examples of files include input and output devices... and named collections of data (maintained on discs or other secondary storage). The creation, manipulation, storage, and retrieval of files is primarily the responsibility of the operating system which establishes the environment in which a BASIC program is executed. Under control of the operating system, a BASIC program can examine and alter the contents of selected files through the use of file input and output statements. Each file has attributes related to the logical representation, organization, and accessibility of its data. In some cases these attributes may reflect a file's physical characteristics, but in general this Standards makes no presumptions concerning the physical representation or organization of files. Thus, unless specifically noted otherwise, all uses of the term 'file' in this Standard refer to the logical interaction of external data with a BASIC program."

Die bei BASIC-Prozessoren verfügbaren Fileanweisungen unterscheiden sich teilweise beträchtlich.[82] Erste Standards für Fileanweisungen enthält ANSI BASIC 79; sie betreffen ausschliessliche sequentiell organisierte Files.[83] Darauf und auf die im Rahmen der FOSBIC Version 2 zulässigen Fileanweisungen sind die folgenden Erläuterungen ausgerichtet. ANSI BASIC 82 weicht bezüglich der Fileanweisungen relativ stark von ANSI BASIC 79 ab.[83a] Die neuen Standardisierungsvorschläge konnten im Rahmen dieser Arbeit nicht vollumfänglich berücksichtigt werden. Besonders wichtig erscheinende Abweichungen gegenüber ANSI BASIC 79 wurden in den Fussnoten vermerkt; verwiesen sei auch auf die Ausführungen in Band 3, Abs. 332.

Bei der FOSBIC Version 2 wird bezüglich der Fileanweisungen auf

- einfache Strukturen,
- klare Abgrenzung der Statementformen gegenüber anderen Anweisungen,
- einfache Wirkungsweise und
- Ausrichtung auf die logischen Beziehungen zwischen Files und BASIC-Programm

besonderen Wert gelegt. Dies gilt auch für die – nach ANSI BASIC noch nicht vorgesehene – relative Fileorganisation mit logisch direktem Zugriff, die gleichzeitig – ohne Veränderung der Fileanweisungen – einen Übergang zur sequentiellen Fileorganisation zulässt.[84]

Die bei der FOSBIC Version 2 verfügbaren Fileanweisungen sind aus Abbildung 6 ersichtlich. Sie gliedern sich in die vier Gruppen der FL-Vereinbarungen, FL-Ausgabeanweisungen, FL-Eingabeanweisungen und FL-Kontrollanweisungen. Daraus ergibt sich die Strukturgleichung

⟨Fileanweisung⟩ := ⟨FL-Vereinbarung⟩ | ⟨FL-Ausgabeanweisung⟩ |
258 206 259 273

⟨FL-Eingabeanweisung⟩ | ⟨FL-Kontrollanweisung⟩
283 288

82 Vgl. Weber und Türschmann, BASIC 2, 1. Aufl., Bern 1977, Abs. 332.4.
83 ANSI BASIC, p. 54–60, 101–104.
83a ANSI BASIC 82, p. 106–149.
84 Zur FOSBIC Version 1 vgl. Weber und Türschmann, BASIC 2, 1. Aufl., S. 99–128.

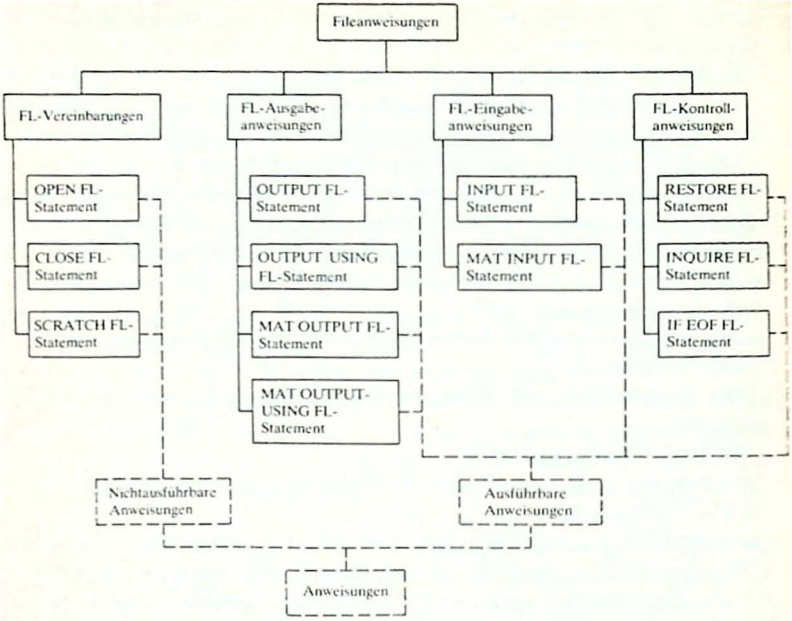


Abbildung 6. Fileanweisungen, FOSBIC Version 2.

Mit den FL-Vereinbarungen werden die Parameter der zu verwendenden Files festgelegt. Sie betreffen Fileorganisation, Filestruktur, maximale Grösse der Sätze, Art der Verarbeitungsmöglichkeiten usw.
Die FL-Vereinbarungen sind den nichtausführbaren Anweisungen zuzurechnen, da durch sie keine programmspezifischen Operationen ausgelöst, sondern vor allem Mitteilungen vom Programm an das auszuführende BASIC-Programmiersystem übergeben werden.
Die FL-Eingabeanweisungen und FL-Ausgabeanweisungen zählen zu den ausführbaren Statements, da durch sie der Datentransfer zwischen dem BASIC-Programm und den benutzereigenen Files gesteuert wird.
Die FL-Kontrollanweisungen dienen zur gezielten Versetzung des File-

pointers und zur Abfrage von Fileparametern, einschliesslich der Pointerposition.[85]

.332.1 *Organisation und Struktur von Files*

Files bestehen aus einer Menge von Sätzen (records). Der inhaltliche Aufbau der Sätze bestimmt die Filestruktur und der logische Aufbau die Fileorganisation.

332.11 Filestruktur

Die Filestruktur zeigt den inhaltlichen Aufbau der Sätze eines Files. Dabei ist zu beachten, dass alle Sätze eines Files die gleiche Struktur haben; eine Mischung von Sätzen unterschiedlicher Struktur ist in einem File nicht möglich.
Nach ANSI BASIC ist zwischen zwei verschiedenen Filetypen zu unterscheiden, die im folgenden sinngemäss zeichen- resp. inhaltstrukturierte File genannt werden.
Bei zeichenstrukturierten Files (display-format files) werden Zeichenfolgen fester oder variabler Länge als Sätze verarbeitet.[86]
Bei den inhaltstrukturierten Files (internal-format files) bestehen die einzelnen Sätze aus einer Folge von numerischen und/oder alphanumerischen Werten.[87]
Falls die Filestruktur nach ANSI BASIC nicht explizit festgelegt ist, gelten die zu verarbeitenden Files implizit als zeichenstrukturiert.[88]

332.111 Zeichenstrukturierte Files

Zeichenstrukturierte Files bauen sich aus Sätzen auf, die aus einer lükkenlosen Folge zulässiger BASIC-Grundsymbolen bestehen.[89] Dabei

85 Zur Erläuterung des Pointers vgl. Abs. 332.111.

86 ANSI BASIC, p. 54: "A display-format file is a sequentially-organized collection of records, where each record is a sequence of characters. For example, the sequence of lines present in response to an input-statement constitutes a file, as does the sequence of lines produced by execution of a print-statement."

87 ANSI BASIC, p. 54: "An internal-format file is a sequentially-organized collection of records, where each record is a sequence of numeric and string values."

88 ANSI BASIC, p. 56; vgl. Abs. 332.2.

89 ANSI BASIC, p. 56: "In a display-format file, each record shall be a string of characters."

entspricht die Satzlänge der Anzahl der in einem Satz enthaltenen Alphazeichen, Ziffern und Sonderzeichen.[90]
Nach ANSI BASIC sind vorläufig nur Files variabler Satzlänge zulässig, wobei die maximale Satzlänge mindestens 72 Zeichen umfassen soll.[91]
Bei zeichenstrukturierten Files kann mithin eine Satzstruktur realisiert werden, die derjenigen einer Ausgabezeile auf dem Bildschirm oder dem Schnelldrucker entspricht.
Sofern ein zeichenstrukturiertes File mit maximaler Satzlänge von 72 Zeichen über die Kanalnummer Null aktiviert wird, erfolgt die Verarbeitung der Fileanweisungen in vollständiger Übereinstimmung mit den elementaren Ein- und Ausgabeanweisungen.[92]
Falls infolge der beschränkten Zeilenkapazität eines Ein-/Ausgabegerätes ein zeichenstrukturierter Satz nicht vollständig in eine Zeile übernommen werden kann, wird der auszugebende Satz üblicherweise – auch bei der FOSBIC Version 2 – der Zeilenstruktur des Gerätes entsprechend gesplittet verarbeitet.
Die bei zeichenstrukturierten Files im Zuge von Schreib- und Lesevorgängen erreichte Position wird durch einen sogenannten Pointer festgehalten.[93] Sein Wert gibt Satz- und Zeichennummer der aktuellen Fileposition an. Fileanweisungen bewirken das Schreiben oder Lesen ab dieser Pointerposition oder eine gezielte Versetzung des Pointers auf diese Position, bevor ein Schreib- und Lesevorgang durchgeführt wird.

332.112 Inhaltstrukturierte Files
Inhaltstrukturierte Files bestehen ebenfalls aus einer Menge lückenlos aufeinander folgender Sätze. Die Satzstruktur wird inhaltsorientiert – nach den zu übernehmenden numerischen und/oder alphanumerischen Werten – festgelegt.

90 ANSI BASIC, p. 57: "The length of a record in a display-format file shall be the number of characters in that record."
91 ANSI BASIC, p. 57: "A record-type specifies the type and maximum length of records in a file. Unless an enhancement to this Standard provides for files composed of fixed-length records, all files shall be composed of variable-length records, i. e., of records whose lengths are independent of each other... The maximum length of these record shall be implementation-defined, but not less than 72 characters per record."
92 Zur Kanalnummer vgl. ANSI BASIC, p. 55: "The file associated with channel zero shall be the external source of data for input-statements and the destination of output from print-statements."
93 ANSI BASIC, p. 56: "A pointer-position specifies the initial position of the pointer for a file."

Nach ANSI BASIC soll ein Satz jeweils nur einen numerischen oder alphanumerischen Wert aufnehmen und dann durch ein entsprechendes Endezeichen abgeschlossen werden.[94]
In der FOSBIC Version 2 kann sich ein inhaltstrukturierter Satz aus mehreren beliebig gemischten Inhalten aufbauen. Die Satzlänge ist variabel und logisch unbegrenzt.
Nach ANSI BASIC erfolgt die Verarbeitung inhalt- und zeichenstrukturierter Files nur bei den Fileausgabeanweisungen unabhängig von der Filestruktur.[95] Bei den Fileeingabeanweisungen besteht zwischen zeichenorientierten Files und dem dafür notwendigen LINPUT-Statement eine Abhängigkeit, die bei der FOSBIC Version 2 aufgehoben ist, indem auch hier das INPUT FL-Statement verwendet wird.[96]
Auch bei inhaltstrukturierten Files wird die im Zuge der Fileverarbeitung erreichte Position durch einen Pointer festgehalten. Die Parameterposition besteht dabei aus dem Tupel Satznummer und Elementnummer[97], wobei diesfalls die Elemente als Inhalte zu verstehen sind.

332.12 Fileorganisation

Die Fileorganisation spezifiziert den logischen Aufbau der Sätze innerhalb eines Files.[98] Nach ANSI BASIC ist als Fileorganisation vorerst nur die sequentielle Aufbauart vorgesehen.[99]
Die FOSBIC Version 2 lässt zusätzlich noch eine zweite Fileaufbauart zu, die als relative Fileorganisation bezeichnet wird.
Die Art der Fileorganisation wird im OPEN FL-Statement[100] festgelegt und kommt in der Folge auch bei den meisten ausführbaren FL-Anweisungen – in den jeweiligen Adressangaben[101] – zum Ausdruck.

94 ANSI BASIC, p. 56: "In an internal-format file, each record shall contain a single numeric or string value. An implementation-defined end-of-records separates records in a file but is not part of a record."
95 ANSI BASIC, p. 101, 103.
96 Vgl. Abs. 332.4.
97 Vgl. S. 136, Gleichung 276.
98 ANSI BASIC, p. 56: "A file-organization specifies the logical organization of records within a file."
99 ANSI BASIC, p. 56: "Unless other organizations are provided in an enhancement of this Standard, only sequential files may be accessed by a program." – Nach ANSI BASIC 82 (p. 111, 115–116) sind vier Organisationsformen vorgesehen: "sequential files, stream files, relative files, keyed files."
100 Vgl. Abs. 332.21.
101 Vgl. Gleichung 275 in Abs. 332.31.

332.121 Sequentielle Fileorganisation

Bei sequentieller Fileorganisation werden die Sätze logisch in linearer Folge angeordnet; sie können nur in dieser Reihenfolge verarbeitet werden.[102]

Die im Zuge der Fileverarbeitung jeweils erreichte Position wird durch den Pointer festgehalten. Durch Schreib- oder Leseanweisungen wird der Pointer bei zeichenstrukturierten Files linear um die entsprechende Anzahl von Zeichen und bei inhaltstrukturierten Files um die entsprechende Anzahl von Inhalten versetzt. Von der jeweils erreichten aktuellen Pointerposition geht jede unmittelbar anschliessende Arbeitsfolge aus; eine direkte Bearbeitung der von der aktuellen Pointerposition nicht referierten Zeichen oder Inhalte ist bei sequentieller Fileorganisation unmöglich.

Die aktuelle Pointerposition hängt von der Struktur der unmittelbar vorangegangenen FL-Anweisung ab. Endet eine FL-Eingabeanweisung ohne Komma, so wird der Pointer nach ihrer Ausführung auf den Anfang des nächsten logischen Satzes gebracht, wobei eventuell Zeichen oder Inhalte des aktuell bearbeiteten Satzes übersprungen werden. Endet eine FL-Eingabeanweisung mit einem Komma, so verbleibt der Pointer – unabhängig vom verarbeiteten Satzumfang – in der Position, die er nach Beendigung der Statementoperation erreicht hat. Damit wird – möglicherweise durch mehrere sich unmittelbar folgende FL-Eingabeanweisungen – eine lückenlose Bearbeitung der Elemente (Zeichen oder Inhalte) des sequentiell organisierten Files erreicht. Folgt andererseits auf eine FL-Eingabeanweisung eine FL-Ausgabeanweisung, so wird nach ANSI BASIC der Pointer vor Ausführung dieser Anweisung an den Anfang des nächsten Satzes vorgerückt.[103] Bei der FOSBIC Version 2 geschieht der Schreibvorgang dagegen ab der aktuellen Pointerposition, wenn die vorausgegangene FL-Eingabeanweisung mit einem Komma endet. Dadurch ist es möglich, auch bei sequentieller Fileorganisation Satzteile innerhalb eines Satzes durch FL-Ausgabeanweisungen zu überschreiben und nicht wie bei ANSI BASIC nur die Änderung ganzer Sätze zuzulassen.

102 ANSI BASIC, p. 56: "A sequential file shall be a linearly ordered sequence of records accessible in sequential order."

103 ANSI BASIC, p. 102: "If the file pointer is not at the beginning of a record following execution of an input-statement or array-input-statement, and if that statement does not end with a final comma, then the pointer shall be advanced to the beginning of the next record in the file. If the statement does end with a comma, then the file pointer shall remain where it was left. A pointer thereby left in the middle of a record shall be advanced to the beginning of the next record before transmission of any output to the file."

332.122 *Relative Fileorganisation*

Bei relativer Fileorganisation werden die Fileelemente relativ zu ihrem Ursprung logisch gespeichert. Als Ursprung der Sätze dient der Fileanfang mit dem Satz- und Elementwert Eins. Die Elemente eines Satzes (Zeichen oder Inhalte) werden relativ zum Satzanfang adressiert.

Die logische Organisation relativer Files entspricht der Elementadressierung bei Matrizen. Die Zeilen entsprechen den Sätzen und die Spalten den Elementpositionen innerhalb eines Satzes.

Nach ANSI BASIC ist eine derartige Fileorganisation noch nicht vorgesehen. Die meisten Minicomputer verfügen jedoch bereits über mehr oder weniger direkte Fileorganisationsarten, wobei – im Gegensatz zur hier definierten relativen Fileorganisation – allerdings keine scharfe Trennung zwischen logischer und physikalischer Fileorganisation vorgenommen wird.[104]

Von der Fileorganisation ist weiterhin die Filebearbeitung zu unterscheiden. Während sequentiell organisierte Files grundsätzlich nur seriell[105] bearbeitet werden können – einzige Ausnahme ist die Zurückversetzung des Pointer durch ein RESTORE-Statement[106] auf den Fileanfang – können relativ organisierte Files sowohl seriell als auch direkt verarbeitet werden. Die Bearbeitung wird dabei entweder durch die Form oder durch die Wertinhalte der Adresse[107] in den FL-Eingabe- oder FL-Ausgabeanweisungen bestimmt. Fehlt die Pointerposition überhaupt oder sind die Wertinhalte der Pointerposition gleich Null, so erfolgt eine serielle Bearbeitung. Die direkte Fileverarbeitung kann allein durch programmablaufbezogene Änderung der Pointerposition auf den Wert Null in eine serielle Fileverarbeitung umgestellt werden.

332.2 *Filevereinbarungen*

Die Filevereinbarungen dienen zur

– Definition eines Files in seiner Organisationart und Satzstruktur, mit Festlegung aller Datenschutzparameter,

104 Vgl. etwa Nixdorf, System-Software Business-Basic, S. 7.45, 57–58, 64–68.

105 Der Begriff seriell ist hier gewählt, um Organisation und Bearbeitung von Files auch begrifflich zu trennen. In der Literatur zur Datenorganisation ist keine eindeutige Verwendung der Begriffe sequentiell und seriell zu finden. Gerhard Niemeyer (Dateiorganisation und -Verarbeitung, München 1975, S. 3–4) spricht zum Beispiel grundsätzlich nur von serieller Speicherung und seriellem Zugriff, während Hartmut Wedekind (Datenorganisation, 3. Aufl., Berlin 1975, S. 52–57) nur die sequentielle Dateiorganisation erwähnt.

106 Vgl. Abs. 332.51.

107 Vgl. Abs. 332.31.

- parameterspezifischen Verbindungsherstellung zwischen Maschinensystem und BASIC-Programm,
- Lösung der Verbindung zwischen BASIC-Programm und Maschinensystem,
- generellen Löschung eines Files, mit allen in der Definition festgelegten Parametern.

Nach ANSI BASIC sind zur Filevereinbarung die folgenden Statements vorgesehen:[108]

$$\underset{259}{\langle \text{FL-Vereinbarung} \rangle} \underset{258}{:=} \left\langle \begin{matrix}\text{OPEN FL-}\\ \text{Statement}\end{matrix} \right\rangle_{260} \mid \left\langle \begin{matrix}\text{CLOSE FL-}\\ \text{Statement}\end{matrix} \right\rangle_{271} \mid \left\langle \begin{matrix}\text{SCRATCH FL-}\\ \text{Statement}\end{matrix} \right\rangle_{272}$$

Diese Statements sind nach ANSI BASIC auf die logische Fileverarbeitung ausgerichtet, was sich bezüglich der Löschanweisung (SCRATCH FL-Statement) darin äussert, dass mit ihm nur der logische Inhalt eines Files – zu dem auch die zur Filedefinition verwendeten Parameter gehören – gelöscht werden kann. Dagegen bewirkt eine logische Filelöschung keine physikalische Dateielemination.

Für die FOSBIC-Version 2 gilt die ANSI BASIC-Norm. Dem gegenüber finden sich bei anderen BASIC-Versionen – insbesondere bei Minicomputersystemen – oft Statementarten, die in gleicher Form auch im Rahmen der Systemsoftware verwendbar sind und – beispielsweise als BUILD- und KILL-Statements – auch zur physikalischen Definition und Löschung einer Datei dienen.[109]

332.21 OPEN FL-Statement

Das OPEN FL-Statement macht eine Datei programmzugänglich.[110] Wird ein OPEN FL-Statement erstmals auf ein File angewendet, so wird damit zunächst die logische Organisation und Struktur definiert;[111] hinzu

108 ANSI BASIC, p. 54–60.
109 Vgl. auch Abs. 332.21, 332.23.
110 ANSI BASIC, p. 54: "Then open-statement makes a file accessible to a program."
111 ANSI BASIC, p. 55: "File-attributes specify logical characteristics of a file and the manner in which a file is to be accessed by a program."

kommen meist einige spezifische Fileparameter, wie Filename und Datenschutzangaben. Die Definition beinhaltet dann auch gleichzeitig die Verbindungsherstellung zwischen BASIC-Programm und Maschinensystem.[112]
Wird das OPEN FL-Statement auf eine physikalische bereits vorhandene Datei ausgeführt, so erfolgt nur die Verbindungsherstellung mit einer Übereinstimmungsprüfung der Fileparameter. Bei Nichtübereinstimmung wird eine Fehlermeldung erzeugt.[113]
Die mit dem OPEN FL-Statement definierten Fileparameter werden vom Computersystem – meist auf dem ersten Satz der Datei – permanent gespeichert.
Nach ANSI BASIC unterliegt das OPEN FL-Statement nachstehender Strukturgleichung; diese trifft auch für die FOSBIC Version 2 zu.

⟨OPEN FL-Statement⟩ :=
260 259

OPEN # ⟨Kanal-nummer⟩ ; ⟨File-name⟩ , ⟨File-parameterliste⟩
261 262 263

Für die Kanalnummer gilt:

⟨Kanalnummer⟩ := ⟨Arithmetischer Ausdruck⟩
261 260 49

112 ANSI BASIC, p. 54: "The open-statement [establishes] the connection between the manner in which a file is identified by the operating system and the manner in which it is identified by a BASIC program. Since the format for identifying files may vary with the operating system, it is assumed only that with each file is associated a string of characters, called its name, which identifies the file to the operating system. A file is identified within a program by the number of a channel through which it is accessed."

113 ANSI BASIC, p. 55: "Execution of an open-statement shall result in a call upon the operating system to provide access to the named file. The interpretation of the value of the file-name is implementation-defined. The index in the channel-expression specifies the channel to which the file is to be assigned; this shall be inactive when the open-statement is executed. If access to the named file is possible in accordance with the file-attribute-list, then the file shall become open and shall be assigned to the specified channel; otherwise an exception shall occur and the channel shall remain inactive."

Nach ANSI BASIC sind nur Kanalnummern zwischen Null und 255 einschliesslich zulässig.[114] Bei der FOSBIC Version 2 ist dieser Wert beliebig; er wird lediglich durch die maximale Grösse der auf der Maschine darstellbaren Ganzzahl begrenzt. Der Wert des arithmetischen Ausdrucks wird normalerweise auf die nächste Ganzzahl gerundet.
Durch die Kanalnummer wird von der Systemsoftware die Verbindung zwischen programmspezifischem File und externer physikalischer Datei geschaffen. Hierzu gehört neben der Reservierung eines Speicherpuffers zur Datenübertragung zwischen Programm und Peripherie auch eine Eintragung in eine im Zentralspeicher geführte Tabelle, die alle notwendigen Verbindungsdaten zwischen logischem File und physikalischer Datei enthält.
Der Filename ergibt sich aus einem alphanumerischen Ausdruck, wobei nach ANSI BASIC das erste Zeichen ein Grossbuchstabe und drei weitere Zeichen entweder Grossbuchstaben oder Ziffern sein sollten;[115] Sonderzeichen und Kleinbuchstaben werden in Filenamen nicht zugelassen. Ausserdem kann der Filename Angaben zur Filesicherung enthalten.[116]

⟨Filename⟩	: =	⟨Alphanumerischer Ausdruck⟩	
262	260		50

Die FOSBIC Version 2 übernimmt als Filename bis zu 20 Zeichen der auf der Maschine darstellbaren BASIC-Grundsymbole;[117] dem Anwender wird allerdings empfohlen, sich an die ANSI BASIC-Norm zu halten, um eine möglichst weitgehende Programmtransportabilität sicherzustellen.
Erwähnt sei, dass nach ANSI BASIC – zusätzlich zum ständig verfügbaren Kanal Null – jedenfalls vier weitere Kanäle gleichzeitig aktiv sein dür-

114 ANSI BASIC, p. 55: "Files are accessed through channels to which they may be assigned during execution of a program-unit; within a program-unit, a channel is identified by a channel number local to that program-unit, which is an integer in the range 0 to 255 inclusive."
115 ANSI BASIC, p. 58: "It is recommended that implementations recognize as file-names strings of characters consisting of an uppercase-letter followed by up to three other uppercase-letters or digits."
116 ANSI BASIC, p. 58: "It is also recommended that information required by the operating system for the purpose of protecting the security of files be considered part of the file-name."
117 Vgl. Bd. 1, Abs. 321.

fen.[118] Die FOSBIC Version 2 begrenzt diese Zahl auf einen implementationsabhängigen Wert, der normalerweise nicht grösser als 10 ist.
Inhaltlich halten sich die meisten übrigen BASIC-Versionen bezüglich des Filenamens an die vorerwähnten Standards; hinsichtlich Struktur und Länge des Namens können allerdings erhebliche Abweichungen auftreten.
Die Fileparameterliste dient zur Festlegung von Filetyp und Fileorganisation sowie zur Bestimmung einer Reihe notwendiger Fileattribute.[119] Die Reihenfolge der Fileattribute innerhalb der Fileparameterliste ist beliebig.

$$\left\langle \begin{matrix}\text{Filepara-}\\ \text{meterliste}\end{matrix} \right\rangle_{263\ 260} := \left\langle \begin{matrix}\text{File-}\\ \text{attribut}\end{matrix} \right\rangle_{264} \mid \left\langle \begin{matrix}\text{File-}\\ \text{parameter}\end{matrix} \right\rangle_{263} , \left\langle \begin{matrix}\text{File-}\\ \text{attribut}\end{matrix} \right\rangle_{264|265}$$

Nach ANSI BASIC gilt für die Fileattribute folgende Strukturgleichung:

$$\langle \text{Fileattribut} \rangle_{264\ 263} := \langle \text{Filetyp} \rangle_{266} \mid \langle \text{Fileorganisation} \rangle_{266} \mid \langle \text{Arbeitsmodus} \rangle_{267} \mid$$

$$\langle \text{Pointerausgangsposition} \rangle_{269} \mid \S\S\S$$

Dabei werden mit §§§ weitere zulässige Fileattribute gekennzeichnet. So kann nach ANSI BASIC zusätzlich der Satztyp spezifiziert werden; bei der FOSBIC Version 2 kann die maschinenspezifische Kanalnummer in die Fileparameterliste übernommen werden.
FOSBIC-spezifisch gilt für die Fileattribute:

$$\langle \text{Fileattribut} \rangle_{265\ 263} := \langle \text{Fileattribut} \rangle_{264} \mid \left\langle \begin{matrix}\text{Maschinenspezifische}\\ \text{Kanalnummer}\end{matrix} \right\rangle_{269}$$

Zur Spezifikation der Fileattribute dienen nach ANSI BASIC spezielle Kennworte; sie sind auch bei der FOSBIC Version 2 verfügbar.

118 ANSI BASIC, p. 58: "The number of channels which may be active simultaneously is implementation-defined. It is recommended that this number be at least four in addition to channel zero." – Zur Differenzierung zwischen geöffneten Dateien und aktiven Kanälen vgl. ANSI BASIC, p. 55: "A file is 'open' if it is currently assigned to a channel and 'closed' otherwise. A channel is 'active' if it currently has some file assigned to it and 'inactive' otherwise."

119 ANSI BASIC, p. 56: "Files opened by execution of an open-statement shall possess the characteristics and be accessible in the manner specified by the file-attribute-list in the open-statement."

Der Filetyp wird durch

⟨Filetyp⟩ : = DISPLAY | INTERNAL
266 264

festgelegt. DISPLAY steht dabei für zeichenstrukturierte und INTERNAL für inhaltstrukturierte Files. Falls kein Filetyp angegeben wird, erfolgt implizit die Öffnung eines zeichenstrukturierten Files.[120]
Die Fileorganisation wird durch das Attribut

⟨Fileorganisation⟩ : = SEQUENTIAL | RELATIVE
267 264

definiert. Nach ANSI BASIC ist nur das Kennwort SEQUENTIAL genormt, obwohl auch noch andere Organisationsformen zukünftig vorgesehen werden.[120a] Das Kennwort RELATIVE dient FOSBIC-spezifisch zur Festlegung der relativen Fileorganisation.[121] Falls eine explizite Angabe zur Fileorganisation fehlt, wird nach ANSI BASIC ein sequentiell organisiertes File geöffnet.[122]
Die FOSBIC Version 2 erlaubt es, ein vorausgehend mit sequentieller Organisation erstelltes File in relativer Organisationsweise zu öffnen und zu bearbeiten. Dadurch besteht zumindest bei der FOSBIC Version 2 eine vollständige Kompatibilität zwischen den verschiedenen Fileorganisationsformen.[123]
Die über Kanalnummer Null permanent zugeschalteten Ein-/Ausgabeeinheiten werden nach ANSI BASIC immer als sequentiell organisierte Files interpretiert.[124]
Der Arbeitsmodus (access-mode) gibt filebezogen die Art der zugelassenen Datenübertragung an.[125] Nach ANSI BASIC gilt:[126]

120 ANSI BASIC, p. 56: "A file-type [DISPLAY, INTERNAL] specifies the logical representation for data in a record of a file ... If no file-type is specified in a open-statement, then the type of the file being opened shall be display-format."
120a Nach ANSI BASIC 82 (p.111–112) sind die Kennworte SEQUENTIAL und STREAM sowie RELATIVE und KEYED vorgesehen.
121 Vgl. Abs. 332.122.
122 ANSI BASIC, p. 56: "If no file organization is specified in an open-statement, then the file-organization of the file being opened shall be sequential."
123 Vgl. hierzu das Beispiel auf S. 141.
124 ANSI BASIC, p. 56: "The external device permanently assigned to channel zero shall be treated as a sequential display-format file."
125 ANSI BASIC, p. 56: "An access-mode specifies the manner in which data in the file shall be accessed."
126 ANSI BASIC, p. 55. – ANSI BASIC 82, p. 111, 114–115.

⟨Arbeitsmodus⟩ : = OUTPUT | INPUT | OUTIN | §§§
268 264

Dabei seien mit §§§ zusätzliche, prozessorspezifische zugelassene Kennworte bezeichnet. Für die FOSBIC Version 2 ist dies INOUT.
Falls als Arbeitsmodus OUTPUT festgelegt wird, können nur Daten vom arbeitenden BASIC-Programm auf das File übertragen werden. Bei INPUT kann nur vom File gelesen werden; es ist nur die Übertragung von Daten aus dem File in das arbeitende BASIC-Programm möglich, so dass ein versehentliches Überschreiben bestehender Fileinhalte ausgeschlossen bleibt. Tritt in der Fileparameterliste das Kennwort OUTIN (oder – FOSBIC-spezifisch – INOUT) auf, so kann sowohl auf das File geschrieben als auch von ihm gelesen werden.[127]
Fehlt die explizite Angabe des Arbeitsmodus, so ist nach ANSI BASIC in der Regel OUTIN zu unterstellen. Für die über Kanal Null zugeschalteten Ein- und Ausgabeeinheiten gilt standardmässig der Arbeitsmodus OUTIN.[128]
Das Fileattribut

⟨Pointerausgangsposition⟩ : = BEGIN | END
269 264

bezieht sich auf die Pointerposition zum Zeitpunkt der Fileöffnung. Durch die Angaben BEGIN wird der Filepointer auf den Anfang, durch END auf das logische Ende des Files positioniert.[129] Die Angabe darf

1127 ANSI BASIC, p. 56: "If the access mode is INPUT, then it shall be possible to read data from the file, but not to write data into it. If the access mode is OUTPUT, then it shall be possible to write data into the file, but not to read data from it. And if the access mode is OUTIN, then it shall be possible both to read data from and to write data into the file."

1128 ANSI BASIC, p. 56: "If no access-mode is specified in an open-statement, then the access-mode of the file being opened shall be OUTIN if the file can both be read and written, INPUT if it can only be read, and OUTPUT if it can only be written. The acess-mode of the external device permanently assigned to channel zero shall be OUTIN."

1129 ANSI BASIC, p. 65–57: "A pointer-position specifies the initial position of the pointer for a file. This pointer indicates the position in the file that will be affected by the next file input or output statement... If the pointer-position is END, then the pointer shall be positioned at the end of the file, i. e., at the position immediately following the last record of the file. If the pointer-position is BEGIN, then the pointer shall be positioned at the beginning of the file, i. e., at the beginning of the first record of the file, or at the end of the file if it contains no records."

nicht im logischen Widerspruch zu den übrigen Fileattributen stehen; so sind insbesondere die Attribute INPUT und END inkompatibel.
Die implizite Festlegung der Pointerausgangsposition erfolgt in Übereinstimmung mit dem gewählten Arbeitsmodus.[130]
Nach ANSI BASIC sollte weiterhin über einen speziellen Fileparameter (record-type) der Satztyp explizit als fix oder variabel definiert[131] und in seiner maximalen Länge bestimmt werden.[132]
Implizit wird mit variabler Satzlänge gearbeitet;[133] bei den über Kanal Null aktivierten I/O-Einheiten kann nur mit variabler Satzlänge gearbeitet werden.[134]
Bei der FOSBIC Version 2 ist die Satzlänge stets variabel und nur durch die Speicherkapazität des benutzten Datenträgers begrenzt. Die nach ANSI BASIC vorgesehene Definition des Satztyps entfällt.
Dagegen ist bei der FOSBIC Version 2 ein spezieller Fileparameter zur Bestimmung der maschinenspezifischen Kanalnummer vorgesehen.

$$\left\langle \begin{matrix}\text{Maschinenspezifische}\\ \text{Kanalnummer}\end{matrix} \right\rangle_{270,\,269} := \text{TAPE}\ \langle\text{Arithmetischer Ausdruck}\rangle_{49}$$

Damit besteht die Möglichkeit zur Trennung zwischen der logischen programminternen und der maschinennotwendigen Kanalnummer.
Der Wert des arithmetischen Ausdrucks wird durch Abschneiden des Dezimalteils ganzzahlig gemacht und muss dann innerhalb eines implementationsabhängigen Intervalls liegen.

130 ANSI BASIC, p. 57: "If no pointer-position is specified in an open-statement, then the pointer-position for the file being opened shall be the beginning of the file if the access-mode for the file is INPUT and the end of the file if the eccess-mode is OUTPUT or OUTIN."
131 ANSI BASIC, p. 57, vgl. Fussnote 91.
132 Zur Zeit ist allerdings nur die variable Form des Record-Typs genormt. Vgl. Abs. 332.111. – Zur Festlegung der variablen Satzlänge gelten nach ANSI BASIC (p. 57) folgende Richtlinien: "The length of a record in a display-format file shall be the number of characters in that record. The length of a record in an internal-format file shall be one if it containes a numeric value; otherwise it shall be the number of characters in the file shall be at most that specified by the record-type in the open-statement opening the file."
133 ANSI BASIC, p. 57.
134 ANSI BASIC, p. 57: "The record-type of the external-device permanently assigned to channel zero shall be variable-length and the maximum length of its records shall be implementation-defined."

Fehlen bei der FOSBIC Version 2 bei Ausführungen des OPEN FL-Statements Fileattributangaben, so werden sie entsprechend der nachstehenden Liste ergänzt.

```
DISPLAY, SEQUENTIAL, OUTIN, BEGIN, TAPE=6
```

Die für die FOSBIC Version 2 realisierte Lösung stimmt damit weitgehend mit den Normvorschlägen von ANSI BASIC überein.
Die meisten Minicomputer trennen Filedefinitionen und Verbindungsherstellung, indem sie hierfür die Anweisungen BUILD (CREATE, DEFINE) und OPEN verwenden. Das OPEN FL-Statement wird dann auf die Struktur

⟨OPEN FL-Statement⟩ : =

OPEN # ⟨Kanalnummer⟩ , ⟨Filename⟩ 261 262

vereinfacht, wobei einige BASIC-Programmiersysteme auch die Fileattribute Arbeitsmodus und Pointerposition in den Angaben des OPEN FL-Statements zulassen. Die übrigen Angaben der Fileparameterliste übernimmt das BUILD-Statement.
Die Aufteilung in BUILD- und OPEN FL-Statement hat auch programmiertechnische Konsequenzen. So kann ein BUILD-Statement nur einmal auf ein File angewendet werden. Alle weiteren Zugriffe müssen durch die Ausführung eines OPEN FL-Statements eingeleitet werden. Teilweise wird die Filedefinition auch nur ausserhalb des BASIC-Programmiersystems zugelassen; das BUILD-Statement ist dann nicht Bestandteil der Programmiersprache BASIC[135] sondern Teil der Systemanweisungen.

135 Dies ist beispielsweise bei der BASIC Version 3 von Control Data der Fall, wobei das BUILD-Statement allerdings durch die Anweisung DEFINE ersetzt ist. Vgl. Control Data. NOS Version 1 Reference Manual. St. Paul 1979, p. 1.8–11. – Bei Nixdorf ist das BUILD-Statement für die Definition von index-sequentiellen Files Bestandteil der System-Software, für die übrigen Fileorganisationsarten dagegen Bestandteil der Programmiersprache BASIC. Vgl. Nixdorf, System-Software Business-Basic, S. 7.48–59.

332.22 CLOSE FL-Statement

Das CLOSE FL-Statement hebt die mit den OPEN FL-Statement für eine bestimmte Kanalnummer hergestellte Verbindung zwischen BASIC-Programm und Maschinensystem wieder auf.[136]
Das CLOSE FL-Statement weist nach ANSI BASIC für alle Filearten folgende Struktur auf:

⟨CLOSE FL-Statement⟩ : =
271 259

CLOSE # ⟨Kanalnummer⟩
261

Das Schliessen eines inaktiven Kanals führt normalerweise zu einer Fehlermeldung.[137]
CLOSE FL-Statements sind in einem BASIC-Programm nur zur vorgezogenen Fileschliessung unbedingt erforderlich. Beim regulären Abschluss der Programmbearbeitung (über ein STOP- oder END-Statement) werden sämtliche Fileverbindungen automatisch geschlossen.[138]
Die FOSBIC Version 2 hält sich an diese Regel. Bei anderen BASIC-Versionen bestehen teilweise unterschiedliche Regelungen, indem nur die logische Beendigung eines Programms über das END-Statement (nicht aber via STOP- oder CHAIN-Statement) zur Schliessung aller Fileverbindungen führt.[139]
Die vorgezogene Fileschliessung via CLOSE FL-Statement kann insbesondere dann erforderlich sein, wenn bei einem BASIC-Programmiersystem die Anzahl der verfügbaren Kanäle begrenzt und ein Kanal sukzessive für verschiedene (nicht gleichzeitig offen zu haltende) Files zu verwenden ist. Dabei besteht auch die Möglichkeit, temporär geschlos-

136 ANSI BASIC, p. 57: "Execution of a close-statement shall close the file assigned to the specified channel, causing the channel to become inactive" – ANSI BASIC 82, p. 118.
137 ANSI BASIC, p. 58.
138 ANSI BASIC, p. 57: "All files still assigned to channels when execution of a program terminates shall be closed."
139 Vgl. Nixdorf, System-Software Business-Basic. S. 10.63.

ssene Files im Zuge der weiteren Programmausführung wiederum zu aktiwieren.[140] Der Kanal Null kann nicht temporär geschlossen werden.
Illm folgenden Beispiel wird zunächst das File ALT mit sequentieller Orgganisation und mit zeichenstrukturierten Sätzen auf der Kanalnummer 8 ggeöffnet. Nach Ausführung des CLOSE FL-Statements mit Statementnnummer 200 wird diese Verbindung gelöst und anschliessend mit dem SStatement 210 sofort wieder für ein relatives File mit dem Namen NEU wund inhaltsstrukturierten Sätzen geöffnet.

```
 10 REM KOMBINATION VON OPEN FL- UND CLOSE FL-STATEMENTS
 20 OPEN #8, "ALT", SEQUENTIAL, BEGIN, DISPLAY, INPUT
 30 INPUT #8, A$
    ...
    ...
200 CLOSE #8
210 OPEN #8, "NEU", RELATIVE, INTERNAL, INOUT, BEGIN
220 OUTPUT #8,3,4: A, B, C
    ...
    ...
900 END
```

332.23 SCRATCH FL-Statement

Das SCRATCH FL-Statement löscht den logischen Inhalt eines Files und versetzt den Pointer an den logischen Fileanfang.[141]
Das SCRATCH FL-Statement weist nach ANSI BASIC folgende Struktur auf:

⟨SCRATCH FL-Statement⟩ : =
272 259

SCRATCH # ⟨Kanalnummer⟩
261

140 ANSI BASIC, p. 57: "It shall be possible to reopen a file closed during execution of a program later during execution of the same program or any of program chained to by the program... without any loss of data in the file."
141 ANSI BASIC, p. 59: "Execution of a scratch-statement shall erase the contents of the file assigned to the specified channel and reset the pointer for that file to the beginning of the file." – Nach ANSI BASIC 82 wird das SCRATCH FL-Statement durch ein ERASE-Statement ersetzt.

Das SCRATCH FL-Statement ist nur auf die im Arbeitsmodus OUTPUT oder OUTIN definierten Files anwendbar;[142] die Verwendung der Kanalnummer Null im SCRATCH FL-Statement bleibt wirkungslos.[143]
Nach Ausführung eines SCRATCH FL-Statements bleibt die Verbindung zwischen BASIC-Programm und Maschinensystem unverändert bestehen; auch der physikalische Fortbestand einer Datei wird vom SCRATCH FL-Statement nicht beeinträchtigt.
Von der FOSBIC Version 2 werden die vorerwähnten Standards voll eingehalten.[144]
Vermerkt sei, dass in ANSI BASIC nicht angegeben wird, inwieweit durch das SCRATCH FL-Statement auch eine Löschung der im OPEN FL-Statement definierten Fileattribute erfolgen kann. Bei der FOSBIC Version 2 erfolgt die logische Löschung eines Files einschliesslich seiner Fileattribute dann, wenn der Ausführung eines SCRATCH FL-Statements logisch unmittelbar ein CLOSE FL-Statement folgt. Vgl. hierzu das folgende Beispiel.

```
1200 REM FILEINHALT LOESCHEN
1210 OPEN #4, "ALT", OUTPUT,SEQUENTIAL,END,INTERNAL
1220  ...
      ...
2000 SCRATCH #4
2010 REM JETZT IST NUR DER FILEINHALT GELOESCHT
2020 REM DIE FILEPARAMETER SIND NOCH VORHANDEN
2030 WRITE #4: A, B, C$
2040  ...
      ...
3000 REM FILEINHALT UND FILEATTRIBUTE LOESCHEN
3010 SCRATCH #4
3020 CLOSE #4
3030  ...
      ...
4000 REM NEUDEFINITION EINES FILES MIT DEM NAMEN -NEU-
4010 REM AUF DERSELBEN KANALNUMMER 4
4020 OPEN #4, "NEU", INOUT,RELATIVE,BEGIN,INTERNAL
4030  ...
      ...
7990 END
```

Bei manchen BASIC-Programmiersystemen tritt das KILL-Statement anstelle der SCRATCH FL-Anweisung. Allerdings wird mit KILL meist

142 ANSI BASIC, p. 60: "[A fatal error occurs if] the file assigned to the channel specified in a scratch-statement cannot be scratched, e. g., because its access-mode specifies input only."
143 ANSI BASIC, p. 59: "If the specified channel is channel zero, then no action shall occur."
144 Das SCRATCH FL-Statement ist auch beim Arbeitsmodus INOUT verwendbar; vgl. Abs. 332.21.

eine logische und physikalische Löschung eines Files durchgeführt und gleichzeitig die entsprechende Kanalnummer desaktiviert. Eine Neudefinition und Eröffnung eines Files hat bei diesen BASIC-Programmiersystemen dann auch meistens durch ein sogenanntes BUILD-Statement zu erfolgen.[145]

332.3 Fileausgabeanweisungen

Fileausgabeanweisungen dienen zur Datenübertragung aus dem Speicherbereich eines BASIC-Programms auf den eines externen Datenträgers.
Die physikalische Verbindungszuordnung der beiden Speicherbereiche erfolgt durch das Betriebssystem; logisch wird die Verbindung entsprechend der im OPEN FL-Statement angegebenen Kanalnummer hergestellt.[146]
Nach ANSI BASIC können vier Fileausgabeanweisungen unterschieden werden.[147]

⟨FL-Ausgabeanweisung⟩ 273 258 : = ⟨OUTPUT FL-Statement⟩ 274 | ⟨OUTPUT USING FL-Statement⟩ 280 |

⟨MAT OUTPUT FL-Statement⟩ 281 | ⟨MAT OUTPUT USING FL-Statement⟩ 282

332.31 OUTPUT FL-Statement

Mit dem OUTPUT FL-Statement können Daten entsprechend den im OPEN FL-Statement festgelegten Angaben auf externe Speicherbereiche übertragen werden.
Das OUTPUT FL-Statement weist generell folgende Struktur auf:

145 Vgl. S. 131.
146 Vgl. Abs. 332.21.
147 ANSI BASIC, p. 101. – Nach ANSI BASIC 82 (p. 126) wird zwischen PRINT FL-, MAT PRINT FL-, WRITE FL- und MAT WRITE FL-Statements differenziert.

⟨OUTPUT FL-Statement⟩ : =
274 273

OUTPUT #	⟨Adresse⟩ 275	:	⟨Outputliste⟩ 279
WRITE #	⟨Adresse⟩ 275	:	⟨Outputliste⟩ 279

Die ersterwähnte Form des OUTPUT FL-Statements ist nach ANSI BASIC vorgesehen;[148] die zweite Form trifft für die FOSBIC Version 2 zu.
Das Kennwort OUTPUT # tritt bei praktisch keinem bekannten BASIC-Compilersystem mit FL-Anweisungen auf. Am häufigsten finden sich die Kennworte WRITE # und PRINT #.
Die zu wählende Adresse hängt wesentlich von der Fileorganisation ab. Sie besteht bei sequentieller Fileorganisation generell aus der Kanalnummer, bei komplexeren Fileorganisationen – beispielsweise bei der nach der FOSBIC Version 2 zulässigen relativen Fileorganisation – kann die Angabe einer Pointerposition hinzukommen.

⟨Adresse⟩ : = ⟨Kanalnummer⟩ | ⟨Kanalnummer⟩ , ⟨Pointerposition⟩
275 274 261 261 276

⟨Pointerposition⟩ : =⟨Satznummer⟩ | ⟨Satznummer⟩ , ⟨Element-nummer⟩
276 275 277 277 278

⟨Satznummer⟩ : = ⟨Arithmetischer Ausdruck⟩
277 276 49

⟨Elementnummer⟩ : = ⟨Arithmetischer Ausdruck⟩
278 276 49

Die sich aufgrund der arithmetischen Ausdrücke für Satz- und Elementnummer ergebenen Werte werden (durch Abschneiden des Dezimalteils) ganzzahlig gemacht.
Fehlt bei der Pointerposition die Elementnummer oder ist ihr Wert gleich Null, so wird der Pointer auf den Ursprung des angegebenen Satzes ge-

148 ANSI BASIC, p. 101.

ssetzt. Weist auch die Satznummer den Wert Null auf oder fehlt sie ebenfalls, so wird – nach der FOSBIC Version 2 – ein relativ organisiertes File logisch seriell verarbeitet.

Eine Kanalnummer Null bewirkt bei zeichenstrukturierten Files eine dem PRINT-Statement vollständig entsprechende Datenübertragung.[149] Analog dazu setzt sich die Outputliste – mit der die zu übertragenden Daten spezifiziert werden – aus der Ausgabe- oder TAB-Ausgabeliste des PRINT-Statements zusammen.[150]

⟨Outputliste⟩ : = ⟨Ausgabeliste⟩ | ⟨TAB-Ausgabeliste⟩ |
279 274 124 127

⟨Outputliste⟩ , ⟨Outputliste⟩ |
279 279

⟨Outputliste⟩ ; ⟨Outputliste⟩
279 279

Bei der Verarbeitung zeichenstrukturierter Files entspricht die Wirkung der in der Outputliste enthaltenen Kommatas und Strichpunkte derjenigen beim PRINT-Statement. Endet eine Outputliste mit einem Komma oder mit einem Semikolon, so wird die Ausgabe des begonnenen Satzes mit dem nächsten logisch folgenden OUTPUT FL-Statement fortgesetzt. Erst wenn eine Outputliste mit einem Leerzeichen endet, wird auch der Satz abgeschlossen und der Pointer auf den Anfang des nächsten Satzes gesetzt.[151] Die Satzlänge eines zeichenstrukturierten Files kann damit theoretisch beliebig gross werden; sie wird nur durch den verfügbaren externen Speicherplatz begrenzt.

149 ANSI BASIC, p. 102: "If output it to channel zero, then [a] string shall be generated exactly as in the execution of a print-statement."

150 ANSI BASIC, p. 101, 68 (print-list). – Zum PRINT-Statement vgl. Bd. 1, Abs. 332.21.

151 Nach ANSI BASIC (p. 103) wird auch der Übergang zu einem anderen Statement geregelt: "When an output-statement which ends with a print-separator is executed, then the last record transmitted to the file may not end with an end-of-record. Subsequently, if input is requested from a file left in this state, or if the file is closed, or if its pointer is reset, then an end-of-record shall be appended to the file before the requested operation is performed."

Die vorerwähnte Verarbeitung trifft auch bei Verwendung der Kanalnummer Null zu; diesfalls werden jedoch Bildschirmzeilen zu jeweils 80 Zeichen generiert, falls ein Satz mehr als 80 Zeichen umfasst.[152]
Wird nach ANSI BASIC ein OUTPUT FL-Statement auf einen bereits vorhandenen Satz ausgeführt und endet die Outputliste ohne Komma bzw. Semikolon oder wird der Arbeitsmodus gewechselt, so geht ein eventueller Satzrestinhalt verloren; der Pointer wird auf den Anfang des nächsten Satzes vorgerückt. Dies auch für den Fall, dass der Arbeitsmodus auf INPUT gewechselt wird. Nach ANSI BASIC geht der frühere Fileinhalt demnach ab der Pointerposition immer verloren, von der aus ein OUTPUT FL-Statement ausgeführt wird.[153]
Bei der FOSBIC Version 2 tritt die vorerwähnte Wirkung nur ein, wenn die Outputliste mit einem Leerzeichen endet. Wird die Outputliste mit einem Komma abgeschlossen, so kommt es nach Ausführung dieses Statements zur Versetzung des Pointers auf die nächste durch 15 teilbare Satzposition; endet die Outputliste mit einem Strichpunkt, so verbleibt der Pointer in der erreichten Position. Ein via OUTPUT FL-Statement eventuell noch nicht überschriebener Satzrest bleibt erhalten und ist auch weiterhin lesbar.
Die FOSBIC-spezifische Verarbeitung zeichenstrukturierter Files zeigt das folgende Beispiel.

```
100 OPEN #1, "BEISPIEL-1", SEQUENTIAL, DISPLAY,BEGIN,INOUT
110 FOR I=1 TO 3
120   WRITE #1: I; ". LOGISCHER SATZ"
130 NEXT I
140 CLOSE #1
150 OPEN #1, "BEISPIEL-1", SEQUENTIAL,DISPLAY,BEGIN,INOUT
160 INPUT #1: B$
170 WRITE #1: "EINGEFUEGT";
180 CLOSE #1
190 END
```

Fileinhalt nach Statement 140:

```
1. LOGISCHER SATZ
2. LOGISCHER SATZ
3. LOGISCHER SATZ
```

152 Falls das entsprechende BASIC-Programm im Batch-Remote-Betrieb ausgeführt wird, erfolgt die Ausgabe bei Kanalnummer Null auf dem Schnelldrucker, wobei die Zeilenlänge normalerweise 132 Zeichen beträgt. Ausserdem existieren Bildschirmterminals mit mehr als 80 Zeichen pro Zeile, so dass diesfalls die Ausgabe in die entsprechende Menge von Bildschirmzeilen unterteilt wird, bis der gesamte logische Satz ausgegeben ist.

153 ANSI BASIC, p. 103: "Output to a file shall be appened to the file starting at the current position of the file's pointer; any data previously in the file following that position shall be lost. When output is complete, the file pointer shall be positioned at the end of the file."

Fileinhalt nach Statement 180:

```
1. LOGISCHER SATZ
EINGEFUEGTER SATZ
3. LOGISCHER SATZ
```

Wird im obigen Beispiel im Statement 170 das Semikolon durch ein Komma ersetzt, so wird der Pointer auf die nächste durch 15 teilbare Position des Satzes nach Ausführung des OUTPUT FL-Statements gesetzt. Diesfalls ist der Fileinhalt nach Statement 180

```
1. LOGISCHER SATZ
EINGEFUEGT      TZ
3. LOGISCHER SATZ
```

Der Satzinhalt ab Position 16 der Zeichenfolge bleibt erhalten. Endet das Statement 170 des obigen Beispiels ohne Semikolon oder Komma, so geht der Restinhalt des zweiten Satzes verloren und das File „BEISPIEL" enthält nach dem Statement 180 den Inhalt

```
1. LOGISCHER SATZ
EINGEFUEGT
3. LOGISCHER SATZ
```

Für inhaltstrukturierte Files setzt sich die Outputliste normalerweise nur aus einer Folge von (durch Kommas getrennten) Ausdrücken zusammen. Jeder Ausdruckswert wird als selbständiger Inhalt gespeichert.
Vermerkt sei, dass nach ANSI BASIC in den sich auf inhaltstrukturierte Files beziehenden Outputlisten auch das Semikolon, die TAB-Funktion und das IMAGE zulässig sind.[154] Dabei geht die übliche Bedeutung des Semikolons allerdings verloren. Komma und Semikolon trennen jetzt gleichbedeutend Inhalte voneinander. Die Wirkung der TAB-Funktion wird unterbunden.[155]
Endet nunmehr eine Outputliste mit einem Komma oder Semikolon, so bleibt der Filepointer auf der Inhaltsposition, die er nach Ausführung des OUTPUT FL-Statements erreicht hat. Ist das letzte Zeichen ein Leerzei-

154 ANSI BASIC, p. 101.
155 ANSI BASIC, p. 103: "If the file assigned to the channel specified in an output-statement... is an internal-format file, then execution of that statement shall generate a sequence of values for transmission to the file, with each value being followed by an end-of-record. Execution of a statement which does not specify an image shall generate the same sequence of numeric and string values whose representations would have been transmitted if output were to a display-format-file; print-separators and tab-calls in the print-list shall have no effect."

chen, so wird der logische Satz abgeschlossen und der Pointer auf den Anfang des nächsten logischen Satzes gesetzt. Eventuell dazwischen liegende Inhalte gehen verloren.
Die für die FOSBIC Version 2 charakteristische Verarbeitung inhaltstrukturierter Files geht aus folgendem Beispiel hervor.

```
100 OPEN #1, "BEISPIEL-2", SEQUENTIAL,OUTPUT,BEGIN,INTERNAL
110 A$ = "INHALT"
120 FOR I=1 TO 3
130   WRITE #1: "SATZ", I,
140   FOR J=1 TO 3
150     WRITE #1: A$, J,
160   NEXT J
170   WRITE #1:
180 NEXT I
190 END
```

Aufgrund dieses Programms ergibt sich nach dem Statement 180 folgender Fileinhalt:

```
SATZ  1  INHALT  1  INHALT  2  INHALT  3
SATZ  2  INHALT  1  INHALT  2  INHALT  3
SATZ  3  INHALT  1  INHALT  2  INHALT  3
```

Jeder logische Satz besteht bei diesem Beispiel aus 8 Inhalten, wovon jeweils vier numerische und vier alphanumerische Werte annehmen. Die Zählweise des Pointers ist inhaltorientiert; er wird pro Satz achtmal um jeweils einen Inhalt versetzt.
Nach ANSI BASIC kann die Adresse im OUTPUT FL-Statement nur ein arithmetischer Ausdruck sein, dessen ganzzahliger Wert die Kanalnummer bestimmt.
Bei der FOSBIC Version 2 wird in der Adresse die Angabe der Pointerposition zugelassen, so dass mit diesen Angaben der Filepointer vor Ausführung des OUTPUT FL-Statements auf die entsprechende logische Zeichen- oder Inhaltposition gesetzt wird.
Eine relative Adressangabe erzeugt immer eine Positionierung des Pointers vor der Statementausführung; dies unabhängig davon, ob in dem logisch vorausgehenden OUTPUT FL-Statement die Outputliste mit einem Komma bzw. Semikolon endet.
Nachstehendes Beispiel zeigt die Anwendung der relativen Fileorganisation bei zeichenstrukturierten Files.

```
100 OPEN #1, "BEISPIEL-1-RELATIV",RELATIVE,BEGIN,DISPLAY
110 FOR I=1 TO 3
120   WRITE #1: I;". LOGISCHER SATZ"
130 NEXT I
140 FOR I=3 TO 1 STEP -1
150   WRITE #1,I,3: "EINGEFUEGT";
160 NEXT I
170 ....
```

Fileinhalt nach Statement 130:

```
1. LOGISCHER SATZ
2. LOGISCHER SATZ
3. LOGISCHER SATZ
```

Fileinhalt nach Statement 160:

```
1.EINGEFUEGT SATZ
2.EINGEFUEGT SATZ
3.EINGEFUEGT SATZ
```

Durch die Statementfolge 140 bis 160 wird im obigen Beispiel in jeden Satz ab der Position 3 der Text „EINGEFUEGT" an die Stelle von „LOGISCHER" geschrieben. Dadurch, dass das Statement 150 mit einem Semikolon endet, geht der Restinhalt des jeweiligen logischen Satzes nicht verloren. Fehlte im Statement 150 das Semikolon, so ergäbe sich folgender Fileinhalt nach Statement 160:

```
1.EINGEFUEGT
2.EINGEFUEGT
3.EINGEFUEGT
```

Wird das Programmbeispiel des Files „BEISPIEL-2" um die Statementfolge 190 bis 270 ergänzt

```
100 OPEN #1, "BEISPIEL-2", SEQUENTIAL,OUTPUT,BEGIN,INTERNAL
110 A$ = "INHALT"
120 FOR I=1 TO 3
130   WRITE #1: "SATZ", I,
140   FOR J=1 TO 3
150     WRITE #1: A$, J,
160   NEXT J
170   WRITE #1:
180 NEXT I
190 CLOSE #1
200 OPEN #2, "BEISPIEL-2", RELATIVE,INTERNAL,BEGIN,OUTPUT
210 FOR I=1 TO 3
220   WRITE #2,I,1: "LOGISCHER SATZ",
230   FOR J=1 TO 3
240     WRITE #2,I,1+2*J: "ELEMENT",
250   NEXT J
260 NEXT I
270 CLOSE #2
280 END
```

so ergibt sich als Fileinhalt:

```
LOGISCHER SATZ  1  ELEMENT  1  ELEMENT  2  ELEMENT  3
LOGISCHER SATZ  2  ELEMENT  1  ELEMENT  2  ELEMENT  3
LOGISCHER SATZ  3  ELEMENT  1  ELEMENT  2  ELEMENT  3
```

Auch hier muss auf die Bedeutung des Kommas am Ende der Anweisungen 220 und 240 hingewiesen werden. Ohne Kommatas würde nach Aus-

führung der jeweiligen OUTPUT FL-Statements auf den Anfang des nächsten logischen Satzes unter Verlust des restlichen Satzinhaltes positioniert. Der Satzinhalt aller drei Sätze wäre diesfalls identisch und hätte folgenden Aufbau:

```
LOGISCHER SATZ  &  ELEMENT  &  ELEMENT  &  ELEMENT
LOGISCHER SATZ  &  ELEMENT  &  ELEMENT  &  ELEMENT
LOGISCHER SATZ  &  ELEMENT  &  ELEMENT  &  ELEMENT
```

wobei das Zeichen & für einen leeren Inhalt steht.
Interessant ist, dass bei einigen BASIC-Programmiersystemen mit Multiprogrammsystemen[156] dem Semikolon oder Komma am Ende der Outputliste eine völlig andersartige Bedeutung zugewiesen wird. Beispielsweise hebt beim System Nixdorf 8870/1 ein Semikolon am Ende der Outputliste die Zugriffssperre auf den gerade bearbeiteten Satz für andere Programme nach Abarbeitung der Outputliste auf.[157] Dadurch wird es programmiertechnisch möglich, die bei Multiprogrammiersystemen auftretenden "Dead lock"-Situationen[158] zu kontrollieren.
Nach ANSI BASIC können bei der Fileausgabe folgende Fehler zu einem Programmabbruch führen:[159]

- Das File wurde unter der angegebenen Kanalnummer nur mit INPUT geöffnet.
- Das File wurde mit der angegebenen Kanalnummer überhaupt nicht geöffnet.
- Die Zeichenlänge einer alphanumerischen Variablen überschreitet die maximale Zeichenlänge eines Satzes.[160]

156 Hierunter wird die gleichzeitige oder ineinander zeitlich verzahnte Verarbeitung mehrerer voneinander unabhängiger BASIC-Programme in einer Zentraleinheit nach festgelegten Prioritäten verstanden; vgl. auch Karl Weber: Elektronische Datenverarbeitung. München 1978, S. 84.

157 Nixdorf: System-Software Business-Basic, S. 7.48–50, 10.80.

158 Unter "Dead lock"-Situation wird eine gegenseitige Blockade von mindestens zwei Programmen beim Zugriff auf mindestens zwei Dateien verstanden, wobei sich die Programme jeweils gegenseitig den Zugriff auf die zu bearbeitenden Dateien sperren. Diese Situation kann nur eintreten, wenn der Zugriff auf die Dateien zeitgleich erfolgt bzw. im Zeitintervall, in dem Dateisperren bestehen.

159 ANSI BASIC, p. 102.

160 Nach ANSI BASIC kann durch eine entsprechende record-type-Angabe im OPEN-Statement die maximale Zeichenzahl eines Satzes festgelegt werden; vgl. ANSI BASIC, p. 55. – Bei der FOSBIC Version 2 ist diese Angabe nicht notwendig, da immer mit variabler Satzlänge gearbeitet wird, wobei die Satzlänge nur durch die Kapazität des externen Speichergerätes begrenzt wird.

- Die Aufnahmekapazität des Files ist erschöpft, bevor die Bearbeitung einer Outputliste abgeschlossen wurde.

332.32 OUTPUT USING FL-Statement

Das OUTPUT USING FL-Statement umfasst, als Erweiterung zum einfachen OUTPUT FL-Statement, auch Formatierungsangaben in Form eines USING FL-Zusatzes.

⟨OUTPUT USING FL-Statement⟩ := 280 273

OUTPUT #	⟨Adresse⟩ 275	⟨Image-Ausgabeliste⟩ 130	
WRITE #	⟨Adresse⟩ 275	⟨Image-Ausgabeliste⟩ 130	

Die ersterwähnte Form ist nach ANSI BASIC vorgesehen; die zweite Form entspricht der FOSBIC Version 2.
ANSI BASIC ist auf sequentiell organisierte Files beschränkt, so dass nur Adressangaben ohne Pointerposition möglich sind. Bei der FOSBIC Version 2 kann das OUTPUT USING FL-Statement auch bei relativer Fileorganisation verwendet werden.[161]
Nach ANSI BASIC soll das OUTPUT USING FL-Statement für ein im DISPLAY-Format geöffnetes File bei Verwendung der Kanalnummer Null dem PRINT USING FL-Statement gleichgesetzt werden.[162]
ANSI BASIC erlaubt die Verwendung des OUTPUT USING FL-Statements auch für Files im INTERNAL-Format.[163] Die dabei erzeugten Inhalte haben ausschliesslich alphanumerischen Charakter. Eine Variablenmaske[164] ergibt jeweils einen alphanumerischen Wert. Ein eventuell

161 Vgl. Abs. 332.12.
162 ANSI BASIC, p. 102–103. – Zum PRINT USING-Statement vgl. Bd. 1, Abs. 332.23.
163 ANSI BASIC, p. 103: "Execution of a statement specifying an image shall generate a sequence of string values corresponding to the formatted output lines, exclusive of end-of-lines, that would have been transmitted if output were to channel zero."
164 Zum Begriff der Variablenmaske vgl. Gleichung 136, Bd. 1, S. 187.

zwischen zwei Variablenmasken liegender Stringinhalt wird ebenfalls als selbständiger Ausgabeinhalt aufgefasst.
Die Wirkung des Kommas und Semikolons entspricht den Regelungen für das einfache OUTPUT FL-Statement. Beide Trennzeichen unterbinden sowohl im DISPLAY- als auch im INTERNAL-Format die Beendigung eines logischen Satzes. Im DISPLAY-Format steuern sie zusätzlich die Anzahl der Leerzeichen, die zwischen zwei aufeinander folgenden Ausgabezeichenketten eingefügt werden.
Das folgende BASIC-Programm baut ein einfaches, zeichenstrukturiertes File zur Speicherung von Konten für verschiedene Kostenarten auf. Es zeigt die gemischte Verwendung von OUTPUT FL- und OUTPUT USING FL-Statements. Obwohl der Fileaufbau meistens sequentiell erfolgt, wird im Statement 210 eine direkte Adressierung durchgeführt, die sicherstellt, dass der Kontoeröffnungswert immer in der Zeichenposition 41 bis 54 steht.
Der erste Satz des Files enthält auf den Positionen 1 bis 4 die Anzahl der gespeicherten Konten.

```
100 DIM T$*30
110 OPEN #3, "KONTEN", RELATIVE,BEGIN,INOUT,DISPLAY
112 IF EOF #3 GOTO 120
114 INPUT #3: T$(1:4)
116 I = VAL(T$(1:4))
118 GOTO 130
120 I = 1
130 PRINT "KONTONUMMER EINGEBEN"
140 INPUT K
150 PRINT "KOSTENARTBEZEICHNUNG EINGEBEN"
160 INPUT T$
170 PRINT "EROEFFNUNGSWERT EINGEBEN"
180 INPUT E
190 WRITE #3,I USING "##########": K;
200 WRITE #3: T$;
210 WRITE #3,I,41 USING "#########.##": E
220 I=I+1
230 PRINT "WEITERE KONTENEINGABE  JA ODER NEIN ? "
240 INPUT T$
250 IF T$ = "JA" GOTO 130
260 IF T$ <> "NEIN" GOTO 230
270 WRITE #3,1 USING "####": I-1
280 CLOSE #3
290 PRINT "PROGRAMM KONTEN ANLEGEN BEENDET, NACHDEM"
300 PRINT I-1;" KONTEN ANGELEGT WURDEN; FILE GESCHLOSSEN"
310 END
```

332.23 MAT OUTPUT FL-Statement

Das MAT OUTPUT FL-Statement unterscheidet sich strukturell geringfügig vom einfachen OUTPUT FL-Statement. Lediglich die Outputliste wird durch die MAT-Ausgabeliste ersetzt.[165]

⟨MAT OUTPUT FL-Statement⟩ : =
281 273

MAT OUTPUT #	⟨Adresse⟩ 275	:	⟨DIM-gebundene / MAT-Ausgabeliste⟩ 254
MAT WRITE #	⟨Adresse⟩ 275	:	⟨MAT-Ausgabeliste⟩ 251

Die ersterwähnte Form ist nach ANSI BASIC vorgesehen; die zweite Form entspricht der FOSBIC Version 2, bei der sowohl DIM-gebundene als auch DIM-begrenzte MAT-Ausgabelisten zulässig sind.
Allgemein ist die Wirkungsweise des MAT OUTPUT FL-Statement im DISPLAY-Format mit der des MAT PRINT-Statements gleichzusetzen; bei sequentieller Fileorganisation und Kanalwert Null ist die Wirkungsweise identisch.
Nach ANSI BASIC soll im DISPLAY-Format am Ende jeder ausgegebenen Matrizenzeile ein end-of-line-Zeichen angefügt werden.[166] Für das MAT OUTPUT FL-Statement bedeutet dies, dass durch eine n-zeilige Matrix n-logische Sätze aufgebaut werden, da das end-of-line-Zeichen diesfalls als end-of-record-Zeichen interpretiert werden soll.[167] Die Fileinhalte beim DISPLAY-Format entsprechen demnach vollständig der normalen Druck- oder Bildschirmausgabe; lediglich die Zeilenlänge kann beliebig gross werden.
Wird ein MAT OUTPUT FL-Statement auf ein File im INTERNAL-Format angewendet, so bewirkt das Semikolon, die gesamte Matrizenausgabe in einem logischen Satz; Komma oder Leerzeichen nach einem Matrizennamen führen zur zeilenweisen Matrizenausgabe mit einer entsprechenden Anzahl logischer Sätze.

165 Vgl. hierzu die Ausführungen beim MAT-Statement, Abs. 331.3, ANSI BASIC, p. 101.
166 ANSI BASIC, p. 99: "An end-of-line [shall be generated] each time a row of the array has been printed."
167 ANSI BASIC, p. 103: "If output is [not to channel zero]... the implementation-defined end-of-line... shall be the end-of-record for the file."

Die FOSBIC Version 2 hält sich an diese Regeln. Bei Adressen mit Pointerposition (im Falle einer relativen Fileorganisation) ist zu beachten, dass durch eine mehrgliedrige MAT-Ausgabeliste in einem MAT OUTPUT FL-Statement entsprechend viele logische Sätze erzeugt werden, auf die direkt zugegriffen werden kann.
Das folgende Beispiel verdeutlicht die FOSBIC-spezifische Wirkungsweise vom MAT OUTPUT FL-Statement. Liegen den Matrizen A, B und C die Dimensionen

```
100 DIM A(5,6), B(9,1), C(17)
```

zugrunde und wird bei relativer Fileorganisation die Anweisung

```
300 MAT WRITE #7,9: A, B; C
```

auf ein File im INTERNAL-Format ausgeführt, so werden (beginnend bei Satznummer 9) durch A fünf logische Sätze mit je 6 Elementen, durch B ein logischer Satz mit 9 Elementen und durch C ebenfalls ein logischer Satz mit 17 Elementen erzeugt. Der Pointer steht nach Ausführung der Anweisung auf dem Anfang des logischen Satzes 16. Wird das Semikolon durch ein Komma ersetzt, so wird für Matrix B eine Folge von neun Sätzen mit jeweils einem Element ausgegeben. Nach vollständiger Ausführung der MAT OUTPUT FL-Anweisung steht der Pointer auf dem Anfang des Satzes 24.
Die meisten Minicomputersysteme lassen eine derartige variable Steuerung der Matrizenausgabe nicht zu; MAT-Ausgabelisten mit Semikolon entfallen. Bei ausreichend grosser Recordlänge wird eine MAT-Ausgabeliste normalerweise in einem physikalischen Satz ausgegeben.
Das nachstehende BASIC-Programm erzeugt ein inhaltstrukturiertes FILE mit dem Namen „EINGABEBEISPIEL" in relativer Organisation, wobei hier die Sätze allerdings sequentiell eingelesen werden.

```
10 DIM A$(5)21
20 OPEN #5, "EINGABEBEISPIEL", BEGIN,OUTPUT,INTERNAL,RELATIVE
30 FOR I=1 TO 5
40   MAT READ A$
50   MAT WRITE #5: A$
60 NEXT I
70 DATA "MIT ", "DER ", "RELATIVEN ", "ADRESSANGABE ", "KOENNEN "
71 DATA "GEZIELT ", "ZEICHEN ", "ODER ", "INHALTE ", "(JE "
72 DATA "NACHDEM ", "OB ", "ES ", "SICH ", "UM "
73 DATA "EIN ", "ZEICHEN- ", "ODER ", "INHALTSTRUKTURIERTES ", "FILE "
74 DATA "HANDELT) ", "IN ", "VARIABLE ", "EINGELESEN ", "WERDEN "
80 CLOSE #5
90 PRINT "FILEAUFBAU BEENDET"
99 END
```

Es entstehen 5 inhaltstrukturierte Sätze mit je 5 alphanumerischen Inhalten.

332.34 MAT OUTPUT USING FL-Statement

Im MAT OUTPUT USING FL-Statement wird die Struktur des einfachen MAT OUTPUT FL-Statements durch das Kennwort USING und die Image-Statementkennung ergänzt.[168]

⟨MAT OUTPUT USING FL-Statement⟩
282 273

MAT OUTPUT #	⟨Adresse⟩ 275	USING	⟨Image-Statement-kennung⟩ 131	: ⟨DIM-gebundene MAT-Ausgabe-liste⟩ 254
MAT WRITE #	⟨Adresse⟩ 275	USING	⟨Image-Statement-kennung⟩ 131	: ⟨MAT-Ausgabe-liste⟩ 251

Die ersterwähnte Form ist nach ANSI BASIC vorgesehen; das Statement ist im DISPLAY-Format bis auf die Adresse identisch mit dem des MAT PRINT USING-Statements.[169] Die zweite Form entspricht der FOSBIC Version 2. Im INTERNAL-Format geschieht der Elementaufbau analog zum OUTPUT USING FL-Statement, während die Steuerung der logischen Satzzahl nach dem MAT OUTPUT-FL-Statement erfolgt; ein Semikolon verursacht die Ausgabe der gesamten Matrix in einem Satz, ein Komma erzeugt einen logischen Satz pro Matrizenzeile.

332.4 Fileeingabeanweisungen

Die Fileeingabeanweisungen führen die Datenübertragung von einem externen Speicherbereich in den Zentralspeicherbereich eines BASIC-Programms durch. Voraussetzung für eine erfolgreiche Datenübertragung ist die logische und physikalische Verbindungsherstellung zwischen

168 ANSI BASIC, p. 101.
169 Vgl. Abs. 331.32.

BASIC-Programm und externem Speicherbereich vor der Ausführung der ersten Fileeingabeanweisung durch ein korrektes OPEN FL-Statement.[170]
Die Fileeingabeanweisungen können generell wie folgt eingeteilt werden:

⟨FL-Eingabeanweisung⟩ (283, 258) := ⟨INPUT FL-Statement⟩ (284) | ⟨MAT INPUT FL-Statement⟩ (286)

Nach ANSI BASIC beziehen sich INPUT und MAT INPUT FL-Statement nur auf Files im INTERNAL-Format; für Files im DISPLAY-Format sind entsprechende LINPUT und MAT LINPUT FL-Statements vorgesehen.[171]
Dieser Differenzierung wird im folgenden nicht gefolgt, da eine Abhängigkeit der Statementform von der Filestruktur logisch nicht erforderlich erscheint.[172]

332.41 INPUT FL-Statement

Die Struktur des INPUT FL-Statements unterscheidet sich vom normalen INPUT-Statement vor allem dadurch, dass anstelle der PROMT-Identifikation die Adresse steht.[173]

⟨INPUT FL-Statement⟩ (284, 283) :=

INPUT #	⟨Adresse⟩ (275)	:	⟨Inputliste⟩ (285)

Nach ANSI BASIC ist die Inputliste nahezu identisch mit der in Band 1 definierten Variablenliste.[174] Der einzige Unterschied besteht nunmehr darin, dass die Inputliste des INPUT FL-Statements mit einem Komma enden kann, was bisher nicht der Fall war.[175]

170 Vgl. Abs. 332.21.
171 ANSI BASIC, p. 101–102. – Nach ANSI BASIC 82 (p. 136) sind folgende Statements vorgesehen: INPUT FL, MAT INPUT FL, LINE INPUT FL, READ FL, MAT READ FL.
172 Zu den Filestrukturen vgl. 332.11.
173 ANSI BASIC, p. 101. – Zum INPUT-Statement vgl. Bd. 1, Abs. 332.14.
174 Vgl. Bd. 1, S. 167.
175 ANSI BASIC, p. 101.

⟨Inputliste⟩	: =	⟨Variablenliste⟩	, |	⟨Variablenliste⟩	
285 284		116		116	

Wird im INPUT FL-Statement als Adresse nur die Kanalnummer spezifiziert, so beginnt die Datenübertragung ab der gegenwärtigen Position des Pointers. Die Daten werden seriell, der Reihenfolge in der Variablenliste entsprechend, verarbeitet. Notwendig ist dabei, dass die Datenart mit der Variablenart korrespondiert; andernfalls erfolgt eine Fehlermeldung.
Endet die Inputliste mit einem Komma, so verbleibt der Pointer in der Position, die er nach der Datenübertragung in die letzte Variable der Inputliste erreicht hat. Fehlt am Ende einer Inputliste das Komma, so wird der Pointer automatisch auf den Anfang des nächsten Satzes gesetzt. Diese Regel gilt solange der Arbeitsmodus INPUT bestehen bleibt.
Wird ein INPUT FL-Statement auf ein zeichenstrukturiertes File ausgeführt, so darf die Variablenliste nur aus alphanumerischen Variablen bestehen. Die Datenübertragung in eine Variable wird so lange fortgeführt, bis entweder das Ende des Satzes oder das Ende des Speicherbereichs der Variablen erreicht ist. Enthält die alphanumerische Variable eine Teilzeichenbegrenzung[176], so werden nur so viele Zeichen aus dem Filesatz übertragen, wie die Teilzeichenbegrenzung zulässt. Folgt einer Stringvariablen ein Komma, so verbleibt der Pointer analog zu den inhaltstrukturierten Files in der Position, die er nach der Datenübertragung in die letzte Variable erreicht hat. Ohne Komma wird der Pointer automatisch auf den Anfang des nächsten Satzes gesetzt.
Beim Wechsel des Arbeitsmodus von INPUT auf OUTPUT oder umgekehrt ist bei der FOSBIC Version 2 darauf zu achten, ob die entsprechenden BASIC-Anweisungen mit einem Komma (bei OUTPUT eventuell auch mit einem Semikolon) enden. Inhalte oder Zeichen können nur verlorengehen, wenn bei der Bearbeitung ein- und desselben Files von OUTPUT auf INPUT gewechselt wird und das zuletzt logisch abgearbeitete OUTPUT FL-Statement mit einem Leerzeichen endet. Ein Wechsel zwischen diesen beiden Statementformen ist nur zulässig, wenn im OPEN FL-Statement als Arbeitsmodus INOUT oder OUTIN angegeben wurde.[176]
Beim Wechsel von INPUT nach OUTPUT, wobei das zuletzt bearbeitete INPUT FL-Statement mit einem Komma endet, sieht ANSI BASIC vor, dass zunächst der Pointer auf den Anfang des nächsten Satzes gesetzt wird, bevor das OUTPUT FL-Statement bearbeitet wird.[177] Damit wird

176 Vgl. Bd. 1, Abs. 323.23.
176 Vgl. Abs. 332.21.
177 ANSI BASIC, p. 102; vgl. S. 122, Fussnote 103.

die bisher eindeutige Logik eines Kommas am Ende eines INPUT-Statements durchbrochen. Ausserdem ist es nicht möglich, Teile eines Satzes mit OUTPUT FL-Statement zu überschreiben.
Bei der FOSBIC Version 2 wird diesbezüglich nicht dem ANSI BASIC-Vorschlag gefolgt, sondern die bisher eindeutige Wirkung eines Kommas am Ende eines INPUT-Statements auch beim Wechsel des Arbeitsmodus beibehalten.
Wird das folgende BASIC-Programm

```
 10 DIM A$(20)21
 20 OPEN #1, "EINGABEBEISPIEL", RELATIVE,INTERNAL,BEGIN,INPUT
 30 INPUT #1,5,2: A$(1), A$(2)
 40 INPUT #1,1,5: A$(3)
 50 INPUT #1,1,1: A$(4), A$(5), A$(6), A$(7)
 60 INPUT #1,2,1: A$(8)
 80 INPUT #1,2,4: A$(9)
 90 INPUT #1,5,3: A$(10), A$(11)
100 MAT PRINT A$(1,7);
110 PRINT
120 PRINT A$(8);A$(9);A$(10);A$(11)
130 CLOSE #1
140 END
```

auf das mit dem Beispielprogramm auf Seite 146 aufgebaute File „EINGABEBEISPIEL" mit dem Inhalt

```
MIT DER RELATIVEN ADRESSANGABE KOENNEN
GEZIELT ZEICHEN ODER INHALTE (JE
NACHDEM OB ES SICH UM
EIN ZEICHEN- ODER INHALTSTRUKTURIERTES FILE
HANDELT) IN VARIABLE EINGELESEN WERDEN
```

angewendet, wobei jede Zeile einem logischen Satz entspricht, so ergibt sich folgende Druckausgabe des Vektors A$:

```
IN VARIABLE KOENNEN MIT DER RELATIVEN ADRESSANGABE
GEZIELT INHALTE EINGELESEN WERDEN
```

Handelt es sich um ein File im DISPLAY-Format, so können mit der relativen Adressangabe durch Satznummer und Pointerposition gezielt Zeichen oder Zeichenketten aus den einzelnen Sätzen herausgegriffen werden. Durch eine Teilzeichenkettenbegrenzung kann die Zeichenübertragung ab der direkt angewählten Position auf eine vorgebbare Anzahl begrenzt werden.
Wurde mit dem Programmbeispiel auf Seite 144 das File „KONTEN" mit aufsteigend sortierten Kontennummern aufgebaut, so vermag das folgende Programm durch Anwendung der sogenannten binären Suchme-

hode[178] mit relativ wenigen direkten Zugriffen die richtige Kontonummer herauszufinden, falls sie vorhanden ist. Nach dem Auffinden einer gesuchten Kontonummer werden Kontobezeichnung und Kontostand gelesen und auf dem Bildschirm ausgegeben. Anschliessend kann durch eine Buchung der Kontostand verändert werden. Der neue Kontostand wird auf das File zurückgeschrieben.

```
 10 DIM A$10,B$12,T$30
 20 OPEN #5, "KONTEN", RELATIVE, INOUT,BEGIN,DISPLAY
 30 INPUT #5: B$(1:4)
 40 N = VAL(B$)
 50 PRINT "KONTONUMMER EINGEBEN"
 60 INPUT K
 70 I1 = 2
 80 I2 = N+1
 90 I = INT(I2-I1)/2+I1
100 INPUT #5,I: A$,
110 K1 = VAL(A$)
120 IF K1 = K GOTO 260
130 IF K1 < K GOTO 170
140 I1 = I+1
150 IF I1 > I2 GOTO 200
160 GOTO 90
170 I2 = I-1
180 IF I2 < 2 GOTO 200
190 GOTO 90
200 PRINT "KONTONUMMER =";K;" NICHT GEFUNDEN !!!"
210 PRINT "MIT BUCHEN FORTFAHREN    JA  ODER  NEIN  ?"
220 INPUT A$
230 IF A$ = "JA" GOTO 50
240 IF A$ = "NEIN" GOTO 380
250 GOTO 210
260 INPUT #5,I,11:T$
270 INPUT #5,I,41:B$
280 W = VAL(B$)
290 PRINT "KONTONUMMER","KONTOBEZEICHNUNG","KONTOSTAND"
300 PRINT K1,T$,B$
310 PRINT
320 PRINT"BUCHUNGSWERT EINGEBEN "
330 INPUT K
340 W = W+K
350 PRINT "NEUER KONTOSTAND =";W
360 WRITE #5,I,41 USING "#########.##": W
370 GOTO 210
380 CLOSE #5
390 PRINT "BUCHUNGSPROGRAMM BEENDET "
400 END
```

Nach ANSI BASIC führen folgende Fehler zu einem Programmabbruch:[180]

– Der Kanal, von dem gelesen werden soll, ist nicht oder nur im OUTPUT-Arbeitsmodus geöffnet.

178 Zur binären Suchmethode vgl. Wedekind, Datenorganisation, S. 170–174.
180 ANSI BASIC, p. 102.

- Variablenart und zu lesender Fileinhalt stimmen nicht überein.
- Die Aufnahmekapazität einer alphanumerischen Variablen reicht zur vollen Übernahme eines alphanumerischen Fileelements innerhalb eines Satzes nicht aus.[181]
- Die Inputliste umfasst mehr Variable, als Daten in einem File vorhanden sind.

332.42 MAT INPUT FL-Statement

Das zur Matrizenverarbeitung dienende MAT INPUT FL-Statement kann folgende Strukturen aufweisen:

⟨MAT INPUT FL-Statement⟩ : =
286 283

MAT INPUT #	⟨Adresse⟩ 275	:	⟨MAT Inputliste⟩ 227a	|
MAT INPUT #	⟨Adresse⟩ 275	:	⟨Erweiterte MAT Inputliste⟩ 287	

Nach ANSI BASIC ist nur die erste Form zulässig.[182] Das nach ANSI BASIC ebenfalls vorgesehene MAT LINPUT FL-Statement wird aus den bereits früher genannten Gründen nicht berücksichtigt.[183]
Aus der vorstehenden Strukturgleichung ist ersichtlich, dass sich die Matrizeneingabeanweisung für Files nur geringfügig vom MAT INPUT-Statement unterscheidet. An die Stelle einer möglichen PROMPT-Identifikation tritt die Adressangabe.[184] Für den Fall, dass die Kanalnummer Null ist, geht die MAT INPUT FL-Anweisung in die normale Form der Dialogeingabe über.

181 Bei vielen BASIC-Programmiersystemen führt dieser Fehler nicht zu einem Programmabbruch. Es wird vielmehr die maximal mögliche Zeichenzahl in die entsprechende Variable übertragen; die restlichen Zeichen bleiben unberücksichtigt. Dieser Verarbeitungsart folgt auch die FOSBIC Version 2.
182 Vgl. ANSI BASIC, p. 101.
183 Vgl. Abs. 332.4, S. 148.
184 Vgl. Abs. 332.14.

Die MAT Inputliste kann im einzelnen neben einem variablen Vektor[185] alle anderen Formen der MAT-Eingabeliste enthalten,[186] wobei die einzelnen Teile durch Kommata voneinander zu trennen sind. Ein Komma am Ende der MAT Inputliste ist nach ANSI BASIC nicht zulässig;[187] die FOSBIC Version 2 hält sich an diese Konvention.
Bei der FOSBIC Version 2 erfolgt die Datenübertragung bei sequentieller Fileorganisation grundsätzlich ab der aktuellen Pointerposition. Bei relativer Fileorganisation kann durch eine entsprechende Adressangabe zunächst der Pointer auf eine bestimmte Satz- und Elementposition gebracht werden, bevor die eigentliche Datenübertragung erfolgt.
Die Elementzahl eines Vektors oder einer Matrix müssen nicht notwendigerweise mit der Elementzahl eines Filesatzes übereinstimmen. Werden für eine Matrixzeile weniger Elemente verlangt, als ein Filesatz enthält, so wird die Datenübertragung in die nächste Zeile ab der jeweils erreichten Pointerposition fortgesetzt; eine Verschiebung des Pointers auf den Anfang des nächsten Satzes erfolgt bei Beginn einer neuen Matrixzeile nicht.[188] Diese Arbeitsweise entspricht exakt der eines normalen MAT INPUT-Statements bei Dateneingabe über ein Terminal.
Die von ANSI BASIC genormte Form der MAT Inputliste gestattet sowohl für das normale MAT INPUT- als auch für das MAT INPUT FL-Statement nur eine rein serielle Datenübernahme in die in der Liste enthaltenen Felder. Für den Dialogbetrieb ergibt sich dabei die Schwierigkeit, dass bei grossen Feldern der Anwender nicht darüber informiert wird, wann bei einer Matrix eine neue Zeile zur Eingabe ansteht.[189] Bei dem MAT INPUT FL-Statement ergibt sich der Nachteil, dass bei Nichtübereinstimmung zwischen der Elementzahl der Sätze und der Elementzahl der Zeilen einer Matrix eine unkontrollierte Datenübertragung erfolgt. Aus diesem Grund soll hier eine Erweiterung der MAT Inputliste vorgeschlagen werden, die mit der FOSBIC Version 2 in ihrer Anwendung getestet wird.

185 Vgl. S. 68, Gleichung 227.
186 Vgl. Gleichung 212.
187 Dies bedeutet, dass nach jedem MAT INPUT FL-Statement der Fileausgangspointer auf einem Satzanfang steht.
188 Vgl. hierzu auch Abs. 332.33. – Bei der MAT OUTPUT FL-Anweisung wird nach Ausgabe jeder Matrixzeile unter bestimmten Bedingungen ein neuer Satz angefangen.
189 Vgl. auch S. 70.

⟨Erweiterte MAT Inputliste⟩ 287 := ⟨MAT Inputliste⟩ 227a 286 |

⟨MAT Inputliste⟩ 227a ; |

⟨MAT Inputliste⟩ 227a ; ⟨Erweiterte MAT Inputliste⟩ 287

Die erweiterte MAT Inputliste unterscheidet sich von der bisher genormten dadurch, dass nach einem Listenelement ein Semikolon (;) erscheinen darf. Die Wirkung des Semikolons verursacht nun eine zeilenorientierte Eingabe sowohl beim MAT INPUT- als auch MAT INPUT FL-Statement, falls es sich bei dem vorausgehenden Listenelement um eine Matrix handelt. Dies bedeutet, dass im Dialogbetrieb für jede neue Matrizenzeile, die Eingabe von einer neuen Bildschirmzeile erfolgt. Der Anwender erhält dadurch eine optische Kontrollmöglichkeit seiner Dateneingabe.
Bei den MAT FL-Statements kann nun für die Matrizeneingabe von einem File ein dem MAT OUTPUT FL-Statement wirkungsanaloges MAT INPUT FL-Statement gegenübergestellt werden. Das Semikolon verursacht, dass mit Beginn der Datenübertragung in eine neue Matrizenzeile der Pointer auf den nächsten Satzanfang positioniert wird. Der Elementinhalt einer derartig eingelesenen Matrix stimmt auch dann mit einer durch MAT OUTPUT FL-Statement ausgegebenen Matrix in den Spalten überein, wenn die Spaltenzahl der Ausgabematrix grösser war als die der Eingabematrix. Lediglich die Inhalte, die in die überzähligen Spalten gehören, werden nicht eingelesen.
Die Wirkungsweise des Semikolons zeigt das nachstehende BASIC-Programm im Vergleich mit der bisherigen Norm.[190]

```
 10 DIM A(5,6), B(4,5), C(4,5)
 20 OPEN #12, "TEST-INPUTLISTE", BEGIN, INTERNAL, SEQUENTIAL, INOUT
 30 L = 0
 40 FOR I=1 TO 5
 50   FOR J=1 TO 6
 60     LET A(I,J) = L
 70     LET L = L+1
 80   NEXT J
 90 NEXT I
100 MAT WRITE #12: A
110 RESTORE #12
120 MAT INPUT #12: B
```

190 Zum RESTORE FL-Statement vgl. Abs. 332.51.

```
1330 RESTORE #12
1440 MAT INPUT #12: C;
1550 CLOSE #12
1660 PRINT "AUF FILE 12 GESCHRIEBENE MATRIX A"
1770 MAT PRINT A;
1880 PRINT " "
1885 PRINT
1990 PRINT "VON FILE 12 MIT NORMALER MAT-INPUTLISTE IN MATRIX B EINGE";
1995 PRINT "LESENE WERTE"
2000 MAT PRINT B;
2010 PRINT " "
2015 PRINT
2020 PRINT "VON FILE 12 MIT ERWEITERTER MAT-INPUTLISTE IN MATRIX C EINGE";
2030 PRINT "LESENE WERTE"
2040 MAT PRINT C;
2050 CLOSE #12
2060 END
```

D0as Ergebnis ist:

```
AUF FILE 12 GESCHRIEBENE MATRIX A
00 1 2 3 4 5
66 7 8 9 10 11
112 13 14 15 16 17
118 19 20 21 22 23
124 25 26 27 28 29

VVON FILE 12 MIT NORMALER MAT-INPUTLISTE IN MATRIX B EINGELESENE WERTE
40 1 2 3 4
55 6 7 8 9
110 11 12 13 14
115 16 17 18 19

VVON FILE 12 MIT ERWEITERTER MAT-INPUTLISTE IN MATRIX C EINGELESENE WERTE
40 1 2 3 4
66 7 8 9 10
112 13 14 15 16
118 19 20 21 22
```

EEin Vorteil der erweiterten MAT Inputliste besteht ausserdem darin, dass die bisherige Norm lediglich eine Erweiterung erfährt. Die Transportfähigkeit von Programmen nach der Norm wird nicht beeinflusst.
EEnthält eine MAT Inputliste einen variablen Vektor, beispielsweise in der nachstehenden Art,

```
120 MAT INPUT #4: A(3,4), B$, V(?)
```

so ist die Anzahl der übertragenen Fileelemente gleich der Zahl, die sich aus der Menge von Elementen zwischen der Pointerposition nach Beendigung der Datenübertragung in den vorausgegangenen Vektor B und dem Ende des Satzes ergibt. Durch Aufruf der parameterlosen Funktion NUM[191] erhält man die Anzahl der übertragenen Elemente des zuletzt

191 Vgl. Abs. 331.12.

eingelesenen variablen Vektors V. Enthält eine MAT Inputliste mehrere variable Vektoren, so kann mit der Funktion NUM jeweils nur die Anzahl der zuletzt gelesenen Fileelemente erfragt werden. Die vorher bearbeiteten variablen Vektoren werden bis an ihre aktuellen Dimensionsgrenzen mit Fileelementen aufgefüllt.
Die Fehlermöglichkeiten des MAT INPUT FL-Statements sind mit denjenigen des INPUT FL-Statements identisch.[192]

322.5 Filekontrollanweisungen

Die Filekontrollanweisungen gestatten
- die Beeinflussung der Pointerposition ohne Datenübertragung (RESTORE FL-Statement),
- die Informationsgewinnung über Art, Name, Organisation, Struktur, Zugriffsmöglichkeit und Pointerposition eines Files (INQUIRE FL-Statement),
- die Kontrolle des logischen und physikalischen Fileendes (IF EOF FL-Statement).

Die Filekontrollanweisungen beziehen sich mithin nicht auf den Fileinhalt; sie erlauben nur die Kontrolle und teilweise auch die Steuerung der Fileparameter, die ein File kennzeichnen und begrenzen.
Für die Filekontrollanweisung gilt die folgende Einteilung:

$$\underset{288}{\langle \text{FL-Kontroll-anweisung} \rangle_{258}} := \underset{289}{\langle \text{RESTORE FL-Statement} \rangle} \mid \underset{290}{\langle \text{INQUIRE FL-Statement} \rangle} \mid \underset{294}{\langle \text{IF EOF FL-Statement} \rangle}$$

Nach ANSI BASIC wird auch das bereits erläuterte SCRATCH FL-Statement zu den FL-Kontrollanweisungen gezählt; da das SCRATCH FL-Statement den Fileinhalt beeinflusst, wird es im Rahmen dieser Arbeit den FL-Vereinbarungen zugeordnet.[193]

192 Vgl. Abs. 332.41.
193 Vgl. Abs. 332.2.

332.51 RESTORE FL-Statement

Durch das RESTORE FL-Statement wird die Position des Filepointers in einem geöffneten File beeinflusst. Die Statementstruktur ist einfach.

⟨RESTORE FL-Statement⟩ 289	288	:=

RESTORE #	⟨Adresse⟩ 275

Nach ANSI BASIC[194] kann als Adresse nur die Kanalnummer angegeben werden; dies deshalb, weil ANSI BASIC sich ausschliesslich auf die sequentielle Fileorganisation bezieht.[195]

Bei sequentieller Fileorganisation wird der FL-Pointer durch ein RESTORE FL-Statement stets auf den logischen Fileanfang zurückgesetzt.[196] Der für ein nachfolgendes INPUT FL-Statement verfügbare Fileumfang wird mithin auf den logischen Gesamtumfang vergrössert. Vgl. Abbildung 7.

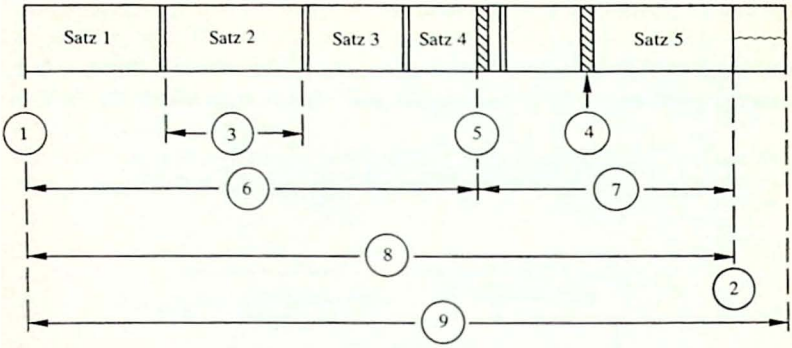
Die Pointerrückversetzung kann beliebig oft wiederholt werden. Eine Verschiebung des logischen FL-Endes ist nur dann möglich, wenn sich der Pointer auf dieser Position befindet und die Filebearbeitung mit einem OUTPUT FL-Statement fortgesetzt wird.[197] Anschliessend kann der erweiterte logische Gesamtumfang eines Files durch ein RESTORE FL-Statement wieder einlesbar gemacht werden.

194 ANSI BASIC, p. 59.

195 Vgl. Abs. 332.11 und 332.31.

196 ANSI BASIC, p. 59: "Execution of a restore-statement with a channel-expression shall reset the pointer for the file assigned to the specified channel to the beginning of the file."

197 Dies unter der Voraussetzung, dass auf dem entsprechenden Speichermedium genügend Speicherkapazität vorhanden ist.



1 = Logischer Fileanfang
2 = Logisches Fileende
3 = Logischer Satzumfang
4 = Datenelement (Zeichen oder Inhalt)
5 = Temporäre Pointerposition
6 = Mit OUTPUT FL- oder INPUT FL-Statement bearbeiteter und mittels RESTORE FL-Statement wieder verfügbar zu machender Fileumfang
7 = Für INPUT FL-Statement verfügbarer Fileumfang vor Ausführung eines RESTORE FL-Statements
8 = Logischer Gesamtfileumfang, für INPUT FL-Statement nach Ausführung eines RESTORE-Statements insgesamt verfügbarer Fileumfang
9 = Physikalische Maximalumfangsmöglichkeit der Datei

Abbildung 7. Wirkungsweise der FL-Statements bei sequentiell organisierten Files – Typ Magnetbanddatei

Für die Anwendung eines RESTORE FL-Statements auf ein sequentiell organisiertes File sei das Beispiel auf S. 138 abgewandelt und gleichzeitig vereinfacht. Die logische Wirkung des Programmabschnitts bleibt dabei vollkommen erhalten.

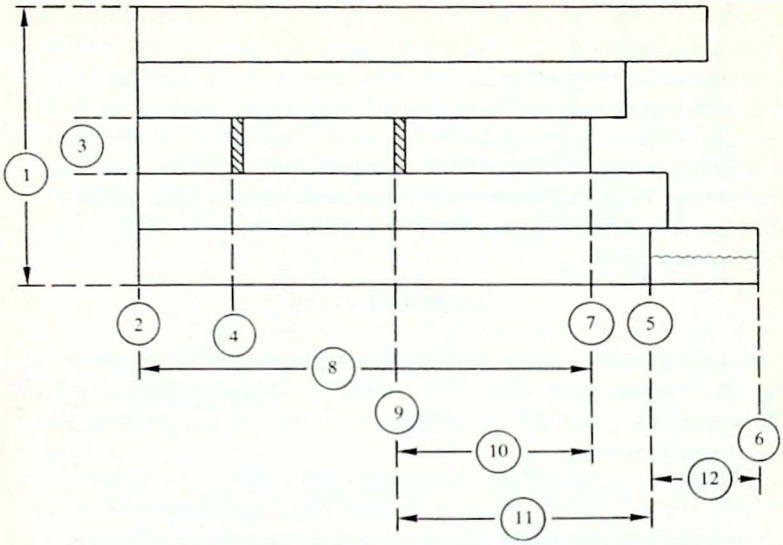
```
100 OPEN #1, "BEISPIEL-1", SEQUENTIAL, DISPLAY, BEGIN,INOUT
110 FOR I=1 TO 3
120   WRITE #1: I; ". LOGISCHER SATZ"
130 NEXT I
140 RESTORE #1
150 REM DIESES STATEMENT ENTFAELLT DURCH DAS RESTORE-STATEMENT
160 INPUT #1: B$
170 WRITE #1: "EINGEFUEGT";
180 CLOSE #1
190 END
```

Bei der FOSBIC Version 2 ist im RESTORE FL-Statement auch die Adressangabe mit Kanal-, Satz- und Elementnummer möglich.[198] Sie bewirkt, dass der Filepointer des entsprechenden Kanals auf das Datenelement des angegebenen Satzes repositioniert wird. Bedingung ist dabei, dass die Satznummer innerhalb des logischen Filegesamtumfanges und die Elementnummer innerhalb der Elementzahl des Satzes liegt; eine Positionierung über das temporäre Fileende hinaus ist nicht möglich.
Fehlt bei der Adressangabe die Elementnummer, etwa nach folgendem Statementbeispiel

```
230 RESTORE #7, 16
```

so wird der Pointer auf den Anfang des angegebenen Satzes gesetzt. Enthält die Adressangabe eines RESTORE FL-Statements für ein relatives File weder Satz- noch Elementnummer, so wird der Filepointer auf den Fileanfang gesetzt.
Abbildung 8 verdeutlicht graphisch die Zusammenwirkung von OUTPUT FL-, INPUT FL- und RESTORE FL-Statements bei relativer Fileorganisation. Satzzahl, Satzlänge und Gesamtfilegrösse werden wiederum nur durch den physikalisch maximal zur Verfügung stehenden Speicherplatz beschränkt. Die Länge der einzelnen Sätze kann unterschiedlich sein.[199]
Wird in Abbildung 8 der Pointer beispielsweise durch ein elementbezogenes RESTORE FL-Statement auf die mit 9 bezeichnete Position im Satz S_i gesetzt, so kann der mit 10 bezeichnete Dateiumfang mit einem logisch folgenden INPUT FL-Statement im Satz S_i gelesen werden. Ebenso können alle Daten in den nachfolgenden Sätzen bis zum logischen Fileende mittels INPUT FL-Statements erfasst werden. Eine Ergänzung des Files durch weitere Sätze im Bereich der mit 12 bezeichneten Position ist nur durch entsprechende OUTPUT FL-Statements möglich. Ebenso können einzelne Sätze in ihrem Datenumfang nur durch entsprechende OUTPUT FL-Statements verkleinert oder vergrössert werden.[200]
Der vorerwähnte Einsatz eines RESTORE FL-Statements ist nur dann notwendig, wenn ein relativ organisiertes File ab einer bestimmten Pointerposition sequentiell bearbeitet werden soll, da andernfalls ein relatives OUTPUT FL-Statement oder INPUT FL-Statement vor der eigentlichen Datenübertragung zunächst die gleiche Wirkung auf die Pointerposition ausübt, wie ein relatives RESTORE FL-Statement. Mithin kann jedes re-

198 Vgl. Abs. 332.31.
199 Vgl. Abs. 332.21, insbesondere S. 130.
200 Vgl. Abs. 332.31.



1 = Satzumfang des Files (n)
2 = Logischer Anfang aller Sätze (S_i, $i=1,2,\ldots n$) des Files; Pointerposition nach sequentiellem RESTORE FL-Statement auf $Satz_1$ bzw. nach satzbezogenem RESTORE FL-Statement auf S_x
3 = Satz S_i
4 = Datenelement D_{ik} ($k = 1,2,\ldots m_i; m_i$ = Elementzahl des Satzes S_i) im Satz S_i
5 = Logisches Ende des Files
6 = Physikalisches Ende der Datei
7 = Logisches Ende des Satzes S_i – Mit WRITE FL-Statement im Satz S_i zuletzt geschriebenes Datenelement D_{im_i}
8 = Logischer Datenumfang des Satzes S_i
9 = Temporäre Pointerposition im Satz S_i nach relativem elementbezogenen RESTORE FL-Statement
10 = Temporärer INPUT FL-Bereich im Satz S_i; ohne neuerliches RESTORE FL-Statement mit INPUT-Statement lesbarer Bereich im Satz S_i
11 = Gesamter temporärer INPUT-Bereich im File – Ohne neuerliches RESTORE FL-Statement im File noch lesbare Datenelemente
12 = Temporärer OUTPUT FL-Bereich für zusätzliche Sätze bzw. Elemente

Abbildung 8. Wirkungsweise der FL-Statements bei relativ organisierten Files – Typ Magnetplattendatei, n Sätze umfassend

lattive OUTPUT FL- oder INPUT FL-Statement durch die Kombination einines relativen RESTORE FL-Statements und eines sequentiellen OUTPUUT FL- resp. INPUT FL-Statements ersetzt werden. So kann beispielsweeise anstelle der Anweisung

```
450 WRITE #7,5,27: A1, B2, C3, X$
```

diie Statementfolge

```
449 RESTORE #7,5,27
450 WRITE #7: A1, B2, C3, X$
```

traeten.

Bei RESTORE FL-Statements bestehen folgende Fehlermöglichkeiten:

- Der angesprochene File ist nicht aktiv (fehlendes OPEN FL-Statement).
- Die angegebene Kanalnummer hat den Wert Null.
- Bei relativer Fileorganisation ist die angegebene Satznummer kleiner als Null oder grösser als die höchstzulässige Satznummer.
- Bei relativer Fileorganisation ist die Elementnummer kleiner als Null oder grösser als die höchste Elementnummer des angegebenen Satzes.

3332.52 INQUIRE FL-Statement

Das INQUIRE FL-Statement dient nach ANSI BASIC zur Abfrage von Fileparametern;[201] bei der FOSBIC Version 2 kann zusätzlich auch die Pointerposition ermittelt werden.

Bezüglich der Statementstruktur gilt:

⟨INQUIRE FL-Statement⟩ : =
290 288

INQUIRE # ⟨Kanalnummer⟩ : ⟨Inquireliste⟩
261 291

Der Aufbau der Inquireliste unterliegt nach ANSI BASIC folgender Struktur: 202)

⟨Inquireliste⟩ : = ⟨Inquireangabe⟩ | ⟨Inquireliste⟩ , ⟨Inquireangabe⟩
291 290 292 291 292

201 ANSI BASIC, p. 59: "Execution of an inquire-statement shall cause the string-variables in the inquiry-items to be assigned values corresponding to the attributes of the file currently assigned to the specified channel."

202 ANSI BASIC, p. 59.

Die Inquireangabe (inquiry-item) setzt sich aus einem Attributnamen und einer Variablen zusammen, wobei die letztere nach ANSI BASIC nur alphanumerischen Charakter aufweisen darf.

⟨Inquireangabe⟩ : = ⟨Attribut⟩ ⟨Einfache alphanumerische Variable⟩ | 45
292 291 293

⟨Attribut⟩ ⟨Einfache numerische Variable⟩ 37|38
293

Nach ANSI BASIC sind die folgenden Attribute zulässig: 203)

⟨Attribut⟩ : = ACCESS | FILE TYPE | NAME | RECORD LENGTH |
293 292 RECORD TYPE | STATUS | §§§

Die Zeichenfolge §§§ steht für zusätzliche compilerspezifische Attribute. Bei der FOSBIC Version 2 sind dies RECORD NUMBER und ELEMENT NUMBER.
Die Bedeutung der einzelnen Attribute ist aus Tabelle 8 ersichtlich.

Tabelle 8. Inquireangaben im INQUIRE FL-Statement

Attribut	Variable	Bedeutung
ACCESS	alphanumerisch	In die Variable wird der Arbeitsmodus (OUTPUT | INPUT | OUTIN) übergeben. Nach der FOSBIC Version 2 ist auch INOUT möglich. Vgl. Gleichung 268.
FILE TYPE	alphanumerisch	Übergabe des im OPEN FL-Statement festgelegte Filetyps (DISPLAY | INTERNAL). Vgl. Gleichung 266.
NAME	alphanumerisch	Übergabe des Filenamens. Vgl. Gleichung 262.
ORGANIZATION	alphanumerisch	Übergabe der Fileorganisationsart (SEQUENTIAL | RELATIVE). Vgl. Gleichung 267; nach ANSI BASIC 79 ist nur SEQUENTIAL genormt.

203 ANSI BASIC, p. 59.

Attribut	Variable	Bedeutung
RECORD LENGTH	alpha-numerisch numerisch	Übergabe der maximalen Satzlänge durch STR$ (N). Bei der FOSBIC Version 2 erfolgt die Übergabe der Elementzahl des Satzes, in dem sich der Pointer aktuell befindet.
RECORD TYPE	alpha-numerisch	Als einzige genormte Form kann nach ANSI BASIC 79 nur VARIABLE als Satztyp übergeben werden. Diese Inquireangabe entfällt bei der FOSBIC Version 2.
STATUS	alpha-numerisch	Nach ANSI BASIC 79 erfolgt die Übergabe eines implementationsabhängigen Werts, der den Zustand der zuletzt auf das File ausgeführten Operation angibt. Bei der FOSBIC Version 2 kann die Angabe INPUT oder OUTPUT erfolgen.
RECORD NUMBER	numerisch	Übergabe der Satznummer, in der sich der Pointer befindet; nur bei der FOSBIC Version 2.
ELEMENT NUMBER	numerisch	Übergabe der Elementnummer, auf die der Pointer positioniert ist; nur bei der FOSBIC Version 2.

Ein nach ANSI BASIC korrektes INQUIRE FL-Statement stellt die nachfolgende Anweisung dar.

```
170 INQUIRE #N: ACCESS A$, NAME B$, STATUS C$
```

Falls ein INQUIRE FL-Statement auf einen inaktiven Kanal ausgeführt wird, soll nach ANSI BASIC in der angegebenen Variablen die Zeichenkette "NONE" erscheinen.[204]
Die einzige Fehlermöglichkeit besteht beim INQUIRE FL-Statement in

[204] ANSI BASIC, p. 59: "If the channel is currently inactive, then all... string-variables [in the inquiry-items] shall receive the value 'NONE'."

der Nichtübereinstimmung der Variablenart in den einzelnen Attributen; bei ANSI BASIC können generell keine numerischen Variablen auftreten.

332.53 IF EOF-Statement

Das IF EOF FL-Statement zählt zu den Programmverzweigungsanweisungen.[205] Es führt bei Auftreten einer EOF-Situation (EOF = End of File) im spezifizierten Kanal zur Ausführung einer Sprunganweisung.

⟨IF EOF FL-Statement⟩			
294	288	: =	

AT EOF #	⟨Kanalnummer⟩	THEN	⟨Statementnummer⟩
	261		71
IF EOF #	⟨Kanalnummer⟩	THEN	⟨Statementnummer⟩
	261		71

Die erste Struktur entspricht ANSI BASIC.[206] Beide Formen werden von der FOSBIC Version 2 akzeptiert.
Die Bedingung des IF EOF FL-Statement trifft zu (TRUE), wenn bei einer FL-Eingabeanweisung die verbliebene Datenmenge nicht ausreicht, um die Inputliste resp. MAT Inputliste aufzufüllen oder wenn durch eine Outputliste einer FL-Ausgabeanweisung das physikalische Ende einer Datei erreicht wird. Diesfalls verzweigt der Programmablauf zu der im IF EOF FL-Statement angegebenen Statementmuster.
Für die Wirkungsweise des IF EOF FL-Statements ist dessen logische Position im BASIC-Programm besonders wichtig. Nach ANSI BASIC muss das IF EOF FL-Statement vor einer eventuellen EOF-Situation durchlaufen sein.[207] Tritt dann in Verbindung mit einer FL-Eingabeanweisung oder FL-Ausgabeanweisung eine EOF-Situation auf, so kehrt der Pro-

205 Vgl. hierzu auch das IF-Statement; Bd. a, Abs. 332.421.
206 ANSI BASIC, p. 101.
207 ANSI BASIC, p. 103: "An end-of-file exception shall occur when not enough data remain in a file to satisfy a request for input from that file or when the physical end of a finite-capacity file is reached before output to that file has been completed."

grammablauf zum logisch vorangegangenen IF EOF FL-Statement zurück und führt die dort angegebene Sprunganweisung aus. Dies bedeutet, dass man mit einem IF EOF FL-Statement am Anfang eines BASIC-Programms grundsätzlich jede EOF-Situation eines Files abfangen kann. Empfehlenswert erscheint es allerdings, die entsprechende IF EOF FL-Anweisung logisch vor das entsprechende FL-Statement zu lokalisieren. Wird vor einer EOF-Situation kein IF EOF FL-Statement erreicht, so kommt es zum Programmabbruch.[208]

Die vorerwähnte ANSI Norm wurde in der Praxis bereits teilweise realisiert. Auch die FOSBIC Version 2 folgt der angeführten Arbeitsweise. Ein korrektes IF EOF FL-Statement wäre demnach die Anweisung

```
270 IF EOF #I THEN 3000
```

nach ANSI BASIC resp.

```
270 AT EOF #I THEN 3000
```

332.6 *Fileanweisungen verschiedener BASIC-Programmiersysteme*

Die im Rahmen von BASIC-Programmiersystemen realisierten Fileanweisungen wurden bisher weitgehend ohne Bezug auf ANSI BASIC entwickelt. Tabelle 9 vermittelt einen Überblick über die bei acht verschiedenen Systemen verfügbaren FL-Anweisungen; eine eingehendere Darstellung dieser Statements findet sich in der ersten Auflage von BASIC 2.[209]

208 ANSI BASIC, p. 103: "If no... exception-statement has been executed for a given channel, then the occurrence of an end-of-file exception for that channel shall result in termination of the program."

209 Weber und Türschmann, BASIC 2, 1. Aufl., S. 129–143.

Prozessor	Öffnen und Definieren von Files	FL-Ausgabe-anweisungen	Positionieren des Datei-zeigers	FL-Eingabe-anweisungen	Schliessen von Files	File-organi-sation[1]	Fileart[2]
ANSI BASIC 79	OPEN #	OUTPUT # OUTPUT USING # MAT OUTPUT # MAT OUTPUT USING #	RESTORE #	INPUT # LINPUT # MAT INPUT # MAT LINPUT #		s	
Dartmouth BASIC-6	FILE # SCRATCH #	WRITE # PRINT # MAT WRITE #	RESET #	INPUT # LINPUT # MAT READ # MAT INPUT # MAT LINPUT # MAT PRINT #		s | r	b | ASCII
UOM BASIC/ UBASIC	OPEN FOR	WRITE ON PRINT ON		INPUT FROM	CLOSE	s | r	b | BCD
IBM CALL BASIC	OPEN	PUT MAT PUT	RESET	GET MAT GET	CLOSE	s | r	BCD
Nixdorf Business-Basic	BUILD # OPEN #	PRINT # WRITE # MAT WRITE #	SEARCH #	READ # MAT READ #	CLOSE #	s | r | i	b | ASCII
FOSBIC Version 1	OPEN COMMON-FILE	PUT MAT PUT	RESET	GET MAT GET	CLOSE	s | i	b

Wang Laboratories	OPEN DATASAVE	PRINT MAT PRINT		READ MAT READ INPUT DATASAVE LOAD	DATASAVE END	s | r | i	b
Digital Computer	ASSIGN CHANNEL	PRINT # MAT PRINT #		INPUT # MAT INPUT #	ICLOSE OCLOSE	r	b | BCD | ASCII
Digital Equipment	OPEN FILE	PRINT PUT MAT PRINT	RESTORE	GET MAT READ	CLOSE	s | r	b | BCD | ASCII

[1] s = sequentiell, r = random, i = index-sequentiell
[2] b = binär

Tabelle 9. Fileanweisungen

333 Erweiterte Programmsteuerungsanweisungen

Die erweiterten Programmsteuerungsanweisungen unterteilen sich in die Subroutine- und Programmsegmentierungsanweisungen. Vgl. Abbildung 9.

⟨Erweiterte Programmsteuerungsanweisungen⟩ 295 := ⟨Subroutineanweisungen⟩ 206 | ⟨Programmsegmentierungsanweisungen⟩ 296 | 315

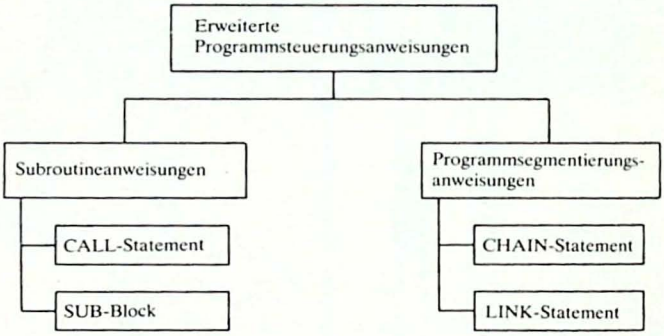


Abbildung 9.

333.1 Subroutineanweisungen

Die Subroutineanweisungen können als spezielle Programmkoppelungsanweisungen charakterisiert werden. Sie gestatten dem BASIC-Anwender den Aufbau und Aufruf eigenständiger Unterprogramme (Subroutine). Dies im Gegensatz zu ANSI Minimal BASIC, das mit einer besonders einfachen Art der Unterprogrammtechnik auskommt.
Nach ANSI Minimal BASIC stehen für die elementare Unterprogrammtechnik GOSUB- und RETURN-Statements zur Verfügung.[210] Bei ihrer Verwendung bleiben die erstellten Unterprogramme Bestandteile des

210 Vgl. Bd. 1, S. 238–244.

gesamten BASIC-Programms; sie stehen durchwegs vor dem END-Statement, so dass die Variablen des Unterprogramms auch Variable des Gesamtprogramms sind. Die bei Abarbeitung eines Unterprogramms bewirkte Wertänderung von Variablen bedeutet für diese eine globale Wertänderung innerhalb des Gesamtprogramms. Jeder Variablenname hat innerhalb des Gesamtprogramms – inklusive Unterprogramm – die gleiche Identität.
Durch die nach ANSI BASIC vorgesehenen Subroutineanweisungen kann der BASIC-Anwender vom übrigen Programmsystem unabhängige Subroutinen aufbauen. An jede Subroutine können über eine Aktualparameterliste Werte übergeben oder von ihr zurückerhalten werden, ohne dass eine Beeinflussung des Wertinhalts anderer – unter dem gleichen Namen in den Subroutinen oder im Hauptprogramm erscheinender – Variabler erfolgt. Die Logik dieser Subroutinetechnik stimmt mit der FORTRAN-Unterprogrammtechnik überein.
Die ANSI BASIC-Norm hat die Vorschläge von NCC Standard BASIC nahezu vollständig übernommen.[211] Für die Subroutineanweisungen gilt:[212]

⟨Subroutine-anweisungen⟩ 296 295 : = ⟨CALL-Statement⟩ 297 | ⟨SUB-Block⟩ 306

333.11 CALL-Statement

Das CALL-Statement dient zum Aufruf einer Subroutine und zur Übergabe von aktuellen Werten und Variablen mittels einer Parameterliste. Die Statementstruktur unterliegt nach ANSI BASIC folgendem Standard.[213]

⟨CALL-Statement⟩ 297 296 : =

CALL	⟨Subroutinename⟩ 298 |		
CALL	⟨Subroutinename⟩ 298	(	⟨Aktualparameterliste⟩ 300)

211 Vgl. Weber und Türschmann, BASIC 2, 1. Aufl., S. 153-157, und die dort angegebene Basisliteratur (NCC Standard BASIC, p. 162–172). – Dies trifft auch für ANSI BASIC 82 (p. 78–81) zu.
212 ANSI BASIC, p. 75. – Vgl. auch Abs. 333.112.
213 ANSI BASIC, p. 75–76.

Der Subroutinename soll aus mindestens einem Grossbuchstaben und höchstens sechs Ziffern oder Grossbuchstaben bestehen, wobei das erste Zeichen immer ein Grossbuchstabe sein muss.[214]

⟨Subroutinename⟩ : = ⟨Grossbuchstabe⟩ |
298 297 3a

⟨Grossbuchstabe⟩ ⟨Zeichenfolge⟩
3a 299

⟨Zeichenfolge⟩ : = ⟨Grossbuchstabe⟩ | ⟨Ziffer⟩ |
299 298 3a 5

⟨Grossbuchstabe⟩ ⟨Zeichenfolge⟩ |
3a 299

Der Subroutinename darf nur einmal vergeben werden und muss mit dem Namen eines korrespondierenden SUB-Blocks übereinstimmen.[215] Weiterhin darf der gewählte Subroutinename nach ANSI BASIC nicht mit Variablennamen übereinstimmen.
Die Aktualparameterliste enthält die durch Komma voneinander getrennten Aktualparameter; sie kann nach ANSI BASIC auch entfallen.[216]

⟨Aktualparameterliste⟩ : = ⟨Aktualparameter⟩ |
300 297 301

⟨Aktualparameter-liste⟩ , ⟨Aktualparameter⟩
300 301

214 ANSI BASIC, p. 75: "Subprogram-names shall contain from one to six characters and shall be distinct from the names of variables."
215 ANSI BASIC, p. 75: "A given subprogram-name shall not occur in more than one sub-statement in a program. A subprogram-name which occurs in a call-statement shall also occur in a sub-statement."
216 ANSI BASIC, p. 75.

Der Aktualparameter kann dazu dienen, einen Wert vom Ausgangsprogramm an die Subroutine zu übergeben. Diesfalls spricht man von einem Aktualparameterwert. Mit einer sogenannten Aktualparametervariablen werden sowohl Werte an die Subroutine übergeben als auch – nach Beendigung der Subroutine – Werte in das aufrufende Programm zurückgegeben. Mithin gilt:

⟨Aktualparameter⟩ (301) := ⟨Aktualparameterwert⟩ (302) |
⟨Aktualparametervariable⟩ (303)

Als Aktualparameter können nur arithmetische und/oder alphanumerische Ausdrücke auftreten.[217] Erscheint eine Variable als Aktualparameterwert, so ist sie nach ANSI BASIC in Klammern zu setzen; damit wird verdeutlicht, dass es sich nicht um eine Aktualparametervariable handelt.[218] Dasselbe gilt auch für Funktionen ohne Parameter.

⟨Aktualparameterwert⟩ (302) := ⟨Arithmetischer Ausdruck⟩ (49) |
⟨Alphanumerischer Ausdruck⟩ (50)

Als Aktualparametervariable können alle einfachen Variablen sowie formale Vektoren und Matrizen auftreten; ausserdem kann ein Kanalparameter angegeben werden. Mithin gilt:

⟨Aktualparametervariable⟩ (303) := ⟨Variable⟩ (35) | ⟨Formaler Vektor⟩ (304) |
⟨Formale Matrix⟩ (305) | ⟨Kanalparameter⟩ (306)

217 ANSI BASIC, p. 77: "A sub-actual-parameter which is an expression, but not a variable, shall be evaluated once on entry to the subprogram and the value so obtained shall be assigned to a location local to the subprogram."

218 ANSI BASIC, p. 77: "A variable in a call-statement may be passed by value by enclosing it in parenthesis to force the sub-actual-parameter to be an expression."

mit

⟨Formaler Vektor⟩ := ⟨Feldname ( ) | ⟨Feldname⟩ $ ( )
304 303 40 40

⟨Formale Matrix⟩ := ⟨Feldname⟩ (,) | ⟨Feldname⟩ $ (,)
305 303 40 40

⟨Kanalparameter⟩ := # ⟨Arithmetischer Ausdruck⟩
306 303 49

Umfasst eine Aktualparameterliste mehrere Parameter, so ist bezüglich Anzahl und Art dieser Parameter vollumfängliche Korrespondenz mit der Formalparameterliste des entsprechenden SUB-Statements[219] erforderlich.[220] Vermerkt sei, dass ein Aktualparameter (mit Ausnahme des Kanalparameters) in einer Aktualparameterliste mehrmals auftreten kann; in der Formalparameterliste des SUB-Statements sollte ein Formalparameter dagegen nur einmal erscheinen, da diesfalls alle gleichen Formalparameter den am weitesten rechts stehenden Aktualparameterwert erhalten.
Ein nach ANSI BASIC korrektes CALL-Statement stellt die Anweisung

```
2330 CALL SORT (A(), B(), (N), L$, #5)
```

dar; dabei sind A( ) und B( ) formale Vektoren numerischen Typs. (N) ein numerischer Ausdruck (Aktualparameterwert), L$ eine alphanumerische Aktualparametervariable und #K ein Kanalparameter.
Formale Fehlermöglichkeiten, die normalerweise zum Programmabbruch führen, sind:
- Nichtexistenz einer Subroutine mit dem aufgerufenen Subroutinenamen.
- Nichtübereinstimmung der Parameteranzahl in den korrespondierenden SUB- und CALL-Statements.
- Nichtübereinstimmung der Parametertypen im korrespondierenden SUB- und CALL-Statement.

Während des Programmablaufs darf sich eine Subroutine weder direkt noch indirekt über eine andere Subroutine selbst aufrufen.[221]

219 Vgl. Abs. 333.112.
220 ANSI BASIC, p. 75: "The number and type of actual parameters in a call-statement shall agree with the number and type of formal-parameters in the corresponding sub-statement."
221 ANSI BASIC, p. 76.

333.12 SUB-BLock

Jede BASIC-Subroutine weist eine spezifische Blockstruktur auf und wird deshalb als SUB-Block bezeichnet.
Ein SUB-Block umfasst mehrere – eigenständig nicht verwendbare – SUB-Statements transitiven Charakters. Sie stellen die Grenzstellen (interfaces) der BASIC-Subroutine dar.
Formal und logisch beginnt ein SUB-Block immer mit einem SUB-Statement; abgeschlossen wird er mit einem SUBEND-Statement. Logisch kann der Programmablauf in einem SUB-Block jederzeit durch ein an beliebiger Stelle in der Subroutine eingefügtes SUBEXIT-Statement aufgegeben werden.[221a]
Die Beendigung einer Subroutine durch ein SUBEXIT- oder SUBEND-Statement führt dazu, dass der Programmablauf auf das dem korrespondierenden CALL-Statement folgendes Statement übergeht,[222] nachdem zuvor eventuell Variablenwerte in die Aktualparameterliste des CALL-Statements von der durchlaufenen Subroutine zurückgegeben werden.
Für die Struktur des SUB-Blocks gilt:

⟨SUB-Block⟩ : =
307 296

⟨SUB-Statement⟩ 308 Folge beliebiger BASIC-Statements, ohne End-Statement, eventuell mit einem oder mehreren ⟨SUBEXIT-Statement⟩ 313 ⟨SUBEND-Statement⟩ 314

Das SUB-Statement ist strukturell identisch mit dem CALL-Statement. Anstelle der Aktualparameterliste steht jetzt die Formalparameterliste.[223]

221a Nach ANSI BASIC 82 (p. 78) sind die Statementformen EXIT SUB und END SUB zu verwenden.
222 Vgl. hierzu auch das RETURN-Statement; Bd. 1. Abs. 332.48.
223 ANSI BASIC, p. 77–78.

⟨SUB-Statement⟩ : =
308 307

SUB ⟨Subroutinename⟩ | 298
SUB ⟨Subroutinename⟩ (⟨Formalparameterliste⟩)   298 309

Die Formalparameterliste kann, wenn sie vorhanden ist, aus einem oder mehreren Formalparametern bestehen, die durch Komma voneinander getrennt sind.

⟨Formalparameterliste⟩ : = ⟨Formalparameter⟩ |
309 308 310

⟨Formalparameter-liste⟩ , ⟨Formal-parameter⟩
309 310

Als Formalparameter dürfen nach ANSI BASIC nur einfache Variable, formale Vektoren bzw. Matrizen oder ein Kanalwert angegeben sein.[224]

⟨Formalparameter⟩ : = ⟨Einfache Variable⟩ |
310 309 311

⟨Formaler Vektor⟩ |
304

⟨Formale Matrix⟩ |
305

⟨Kanalwert⟩
312

224 ANSI BASIC, p. 75.

Der Typ der einfachen Variablen kann numerisch oder alphanumerisch sein, muss aber mit dem des Aktualparameters im CALL-Statement übereinstimmen.

⟨Einfache Variable⟩ : = ⟨Einfache numerische Variable⟩ |
311 310 37

⟨Einfache alphanumerische Variable⟩
45

Der Kanalwert korrespondiert strukturell mit dem Kanalparameter des CALL-Statements. Anstelle des arithmetischen Ausdrucks kann aber nur eine positive Ganzzahl stehen.[225]

⟨Kanalwert⟩ : = # ⟨I-formatierte Ganzzahl, grösser Null⟩
312 310 22

Tabelle 10 enthält eine Korrespondenzliste für Aktual- und Formalparameter in CALL- und SUB-Statements.[226] Daraus ist ersichtlich, dass für formale Vektoren und Matrizen stets eine zweiseitige Beziehung besteht, indem der gesamte Vektor bzw. die gesamte Matrix mit den bei Ausführung des CALL-Statements bestehenden Dimensionen an die Subroutine übergeben und von dort – eventuell sogar redimensioniert – zurückgenommen wird.[227] Dasselbe gilt für Variable, mit denen allerdings nur ein einziger Wert an die Subroutine übergeben wird und von dort zurückgelangt.

225 ANSI BASIC, p. 76: "Channel number zero shall not appear as a sub-formal-parameter."

226 ANSI BASIC, p. 75–76: "The number and type of actual parameters in a call-statement shall agree with the number an type of formal-parameters in the corresponding sub-statement. A formal-array used as a sub-actual-parameter shall have the same number of dimensions as the corresponding sub-formal-parameter. A given sub-formal-parameter shall occur only in a sub-statement."

227 ANSI BASIC, p. 77: "References within a subprogram to sub-actual-parameters which are formal-arrays shall result in references to the corresponding arrays in the sub-actual-list; assignments to or redimensioning of such arrays shall result in assignments to or redimensioning of the corresponding arrays in the sub-actual-list."

Erstmalige Dimensionierungen und Stringlängenfestlegungen sind in der Subroutine nicht zulässig.[228] Falls in der Subroutine ein Vektor- oder Matrixelement als Variable auftritt und der Vektor bzw. die Matrix redimensioniert wird, soll das Ergebnis nach ANSI BASIC implementationsabhängig sein.[229]
Wird durch ein CALL-Statement ein Kanalparameter an eine Subroutine übergeben, so braucht der Kanal zum Zeitpunkt der Ausführung des CALL-Statements nicht unbedingt geöffnet zu sein.[230] Für bereits aktive und in der Subroutine geöffnete Files gelten folgende Regelungen:[231]

- Der errechnete Kanalwert soll innerhalb der Subroutine benutzt werden.
- Die Fileparameter sollen unverändert übergeben werden.
- Änderungen der Fileparameter in der Subroutine sollen an das aufrufende Programm zurückgelangen.

Eine einseitige Beziehung zwischen aufrufendem Programm und Subroutine tritt auf, wenn als Aktualparameter ein Ausdruck angegeben wird.

Bedeutung	Aktualparameter	Formalparameter	Übergabeart
Parameteranzahl	n	n	
Parametertyp	numerisch alphanumerisch	numerisch alphanumerisch	
Parameterart	Formaler Vektor Formale Matrix Variable Kanalparameter Ausdruck	Formaler Vektor Formale Matrix einfache Variable Kanalwert einfache Variable	zweiseitig zweiseitig zweiseitig zweiseitig einseitig

Tabelle 10. Korrespondenz zwischen Aktual- und Formalparametern

228 ANSI BASIC, p. 76: "An array-name which is a sub-formal-parameter shall not appear in a dimension-statement within the subprogram. A string-variable which is a sub-format-parameter shall not occur with a length-fixer in a dimension-statement within the subprogram."

229 ANSI BASIC, p. 77: "If both an array and one of its elements are used as sub-actual-parameters in a call-statement and the array is redimensioned during execution of the subprogram, then any subsequent reference within the subprogram to the formal parameter associated with the array-element shall produce implementation-defined results."

230 ANSI BASIC, p. 77: "A file need not be assigned to a channel designated by a sub-actual-parameter when a call-statement is executed."

Der berechnete Ausdruckswert wird an eine einfache Variable gleichen Typs der Formalparameterliste übergeben. Wertänderungen dieser einfachen Variablen im Subroutineablauf wirken sich nicht im Ausdruck des rufenden Programms aus.
Das SUBEXIT-Statement besteht generell nur aus dem Kennwort SUBEXIT.

SUBEXIT-Statement : =
313 307

SUBEXIT

Es kann im Rahmen eines SUB-Blocks eigenständig oder in Verbindung mit einem IF-Statement auftreten.[232]
Das SUBEND-Statement schliesst einen SUB-Block ab.[233]

SUBEND-Statement : =
314 307

SUBEND

Die folgende Anweisungsfolge stellt ein Beispiel für einen korrekt aufgebauten SUB-Block dar.

```
2000 SUB SEARCH (A(), X, N, I)
2010 REM LINEARES SUCHEN EINES ZAHLENWERTES IM VEKTOR A
2020 FOR I=1 TO N
2030   IF A(I) = X THEN SUBEXIT
2040 NEXT I
2050 I=0
2060 SUBEND
```

231 ANSI BASIC, p. 77: "A sub-actual-parameter which is a channel-expression shall be evaluated once on entry to the subprogram and the resulting channel shall be used whenever the corresponding sub-formal-parameter is referenced in the subprogram. The properties of the file assigned to this channel shall be passed unchanged to the subprogram, and changes to them within the subprogram shall remain in effect upon exit from the subprogram."
232 ANSI BASIC, p. 75.
233 ANSI BASIC, p. 76: "The subend-statement serves both to mark the textual end of a subprogram and, when executed, to terminate execution of the subprogram."

Das Subprogramm sucht aus einem Vektor A linear den Wert X heraus und gibt über I dessen Position innerhalb des Vektors an, falls X vorhanden ist. Andernfalls wird I auf Null gesetzt.
Bemerkenswert ist beim vorstehenden Programm, dass alle vorkommenden lokalen Variablennamen A, X, N und I in der Übergabeart zweiseitig sind. Eine Wertänderung in diesen Variablen führt beim Rücksprung in das aufrufende Programm zur automatischen Übergabe dieser Werte an die Aktualparametervariablen des CALL-Statements.
Vermerkt sei, dass der Geltungsbereich der zur Kennzeichnung von Formalparametern, Variablen und benutzerdefinierten Funktionen verwendeten Namen lokal auf die betreffende Subroutine begrenzt ist. In anderen Subroutinen oder im Hauptprogramm verwendete Variable oder benutzerdefinierte Funktionen gleichen Namens sind vollständig voneinander unabhängig.
Vektoren und Matrizen, die nicht in der Formalparameterliste vorkommen, können in der Subroutine zusätzlich dimensioniert werden.[234] Fileangaben, die sich nicht auf die Kanalnummer Null oder auf den Formalparameter beziehen, sind *lokale* Grössen der Subroutine. Diese Files müssen durch ein entsprechendes OPEN FL-Statement[235] geöffnet und sollen vor dem Verlassen der Subroutine geschlossen werden.[236] DATA-, READ- und RESTORE-Statements[237] innerhalb der Subroutine sind in ihrer Wirkung ebenfalls lokal auf diese begrenzt.
Dies bedeutet, dass der Pointer des aus den DATA-Statements der aufgerufenen Subroutine aufgebauten Files[238] vor Ablaufbeginn jeweils auf

234 ANSI BASIC, p. 78: "Variables and arrays which are not subformal-parameters shall be initialized or not initialized in a manner consistent with the implementation policies each time the subprogram is called. Arrays in a subprogram which are not sub-formal-parameters shall appear in a dimension-statement in the subprogram if they are to have other than the default dimension."
235 Vgl. Abs. 332.22.
236 ANSI BASIC, p. 78: "Nonzero channel numbers within a subprogram which are not sub-formal-parameters shall designate channels local to the subprogram, which shall be distrinct from channels designated by the same number outside the subprogram. Channel zero shall be global to the entire program. Files shall be assigned to nonzero channels within a program-unit by means of an open-statement before use. Files assigned to channels local to a subprogram shall be closed upon exit from the subprogram."
237 Vgl. Bd. 1, Abs. 332.1.
238 Vgl. Bd. 1, Abs. 332.11. – Demnach wird für jede Subroutine, die DATA-Statements enthält, ein individuelles File aufgebaut.

den Fileanfang gesetzt wird.[239] Ebenfalls lokal auf die Subroutine beschränkt sind die dort benutzerdefinierten Funktionen[240] und ein eventuelles OPTION-Statement.[241] Dagegen wird durch den eingebauten Zufallszahlengenerator für alle Programme eine globale Folge von Zufallszahlen erzeugt.[242]

Eine BASIC-Subroutine stellt mithin ein nahezu selbständiges BASIC-Programm dar. Eine Verbindungsstelle zum aufrufenden Programm wird nur durch das SUB-Statement definiert, über dessen Formalparameterliste sich die Programme gegenseitig beeinflussen können.

Das folgende BASIC-Programm ist eine geänderte Version des auf Seite 43 aufgeführten Programms zur Lösung linearer Gleichungssysteme. Der eigentliche Lösungsalgorithmus ist vom Hauptprogramm getrennt und in eine Subroutine verlagert worden.

```
  5 REM LOESUNG LINEARER GLEICHUNGSSYSTEME MIT SUBROUTIEN LINEQU
 10 DIM A(10,10), B(10)
 15 X9 = 10
 20 PRINT "ANZAHL DER GLEICHUNGEN DES SYSTEMS EINGEBEN"
 30 INPUT N
 40 IF N <= X9 GOTO 70
 50 PRINT "ANZAHL DER GLEICHUNGEN GROESSER ALS ";X9;" NICHT ERLAUBT"
 55 PRINT "DIMENSION VON A UND B ERHOEHEN"
 60 GOTO 220
 70 MAT A = ZER(N,N+1)
 80 MAT B = ZER(N)
 90 PRINT "KOEFFIZIENTENMATRIX ZEILENWEISE EINGEBEN"
100 MAT INPUT A(N,N)
110 PRINT "LINKE SEITE DES GLEICHUNGSSYSTEM EINGEBEN"
120 MAT INPUT B(N)
130 P=0
140 CALL LINEQU(A(,), B(), N, P)
150 IF P=0 GOTO 180
160 PRINT "KOEFFIZIENTENMATRIX DES GLEICHUNGSSYSTEMS IST SINGULAER"
170 GOTO 220
180 PRINT "LOESUNG DES GLEICHUNGSSYSTEMS"
190 FOR I=1 TO N
200   PRINT "X(";I;")=";B(I)
210 NEXT I
220 PRINT "NEUE DATENEINGABE GEWUENSCHT  JA ODER NEIN ?"
230 INPUT F$
```

239 ANSI BASIC, p. 78: "Data within a subprogram shall be local to that subprogram. Hence, read- and restore-statement within a subprogram shall refer only to the data in data-statements within that subprogram and not to data in the main program. Upon entry to a subprogram, pointers to the data within that subprogram shall be reset to the beginning of the data."

240 ANSI BASIC, p. 78: "An option-statement within a subprogram shall be local to that subprogram but shall affect all arrays referenced with that subprogram, including those which are parameters to the subprogram."

241 ANSI BASIC, p. 78: "The random number sequence is not local, but rather global to the main-program and all subprograms."

```
 240 IF F$ = "JA" GOTO 20
 250 IF F$ = "NEIN" GOTO 270
 260 GOTO 220
 270 PRINT "PROGRAMM ZUR LOESUNG LINEARER GLEICHUNGSSYSTEME BEENDET"
 280 END
1000 SUB LINEQU(F(,), X(), N1, B)
1002 DIM D(10,11)
1004 MAT D = ZER(N1,N1+1)
1006 MAT D = F
1010 LET J = N1+1
1020 FOR I=1 TO N1
1030   LET D(I,J)=X(I)
1040 NEXT I
1050 REM LOESUNGSALGORITHMUS
1380 FOR M1=1 TO N1
1390   LET T=D(M1,N1)
1400   IF T = 0 GOTO 1550
1410   FOR N=1 TO J
1420     LET D(M1,N)=D(M1,N)/T
1430 NEXT N
1440   LET M = 1
1450   IF M = M1 GOTO 1510
1460   LET T = D(M,M1)
1470   FOR N=1 TO J
1480     LET D(M,N) = D(M,N)-D(M1,N)*T
1490   NEXT N
1510   LET M = M+1
1520   IF M <= N1 GOTO 1450
1530 NEXT M1
1532 FOR N=1 TO N1
1534   X(N) = D(N,J)
1536 NEXT N
1540 SUBEXIT
1550 REM KEINE ZULAESSIGE LOESUNG, MATRIX IST SINGULAER
1560 LET B = 1
1570 SUBEND
```

Aus dem vorstehenden Programmbeispiel ist auch die von ANSI BASIC vorgeschlagene Reihenfolge von Haupt- und Subroutinen ersichtlich. Letztere folgen grundsätzlich dem END-Statement des Hauptprogramms.[242] Die Anzahl der an ein Hauptprogramm anfügbaren Subroutinen wird bei allen Systemen durch die beschränkte Speicherkapazität auf einen implementationsabhängigen Wert begrenzt.
Bei Verwendung von CALL-Statements und entsprechenden SUB-Blökken, entspricht der logische Programmablauf bis auf die Behandlung der Variablen, der Unterprogrammtechnik mit GOSUB- und RETURN-Statements.[243]

242 ANSI BASIC, p. 12: "A BASIC-program is divided logically in a number of program-units. The first of these is the main program, which is terminated by an end-line. Following the main-program may be zero or more subprograms or picture-definitions, each of which is a separate program-unit."
243 Vgl. Bd. 1, Abs. 332.49.

Da in einer BASIC-Subroutine wiederum ein CALL-Statement auftreten darf, können Subroutinen ineinander geschachtelt werden. Die Anzahl der CALL-Statements, die vor einem SUBEXIT- oder SUBEND-Statement ausgeführt werden, bestimmen die maximale Schachtelungstiefe; diese ist implementationsabhängig und dürfte bei den meisten Systemen um 20 liegen.

Bei der Verschachtelung mehrerer Subroutinen muss sichergestellt sein, dass eine bereits aufgerufene Subroutine nicht noch einmal aufgerufen wird.[244] Ohne diese Restriktion könnten unbegrenzte Aufrufschleifen entstehen.

Die Fehlermöglichkeiten bei BASIC-Subroutinen bestehen zunächst hauptsächlich darin, dass Formvorschriften bezüglich des SUB-Blocks verletzt werden. Ansonsten müssen für die übrigen BASIC-Anweisungen innerhalb des SUB-Blocks dieselben Vorschriften eingehalten werden, wie sie für die einzelnen Statements festgelegt wurden.

Wird ein formaler oder logischer Fehler erst während des Programmablaufs in einer Subroutine festgestellt, so erfolgt der Programmabbruch dort, ohne dass ein Rücksprung in das aufrufende Programm stattfindet.

333.2 *Programmsegmentierungsanweisungen*

Programmsegmentierungsanweisungen dienen dazu, ein laufendes BASIC-Programm an einer beliebigen Stelle zu beenden und ein anderes BASIC-Programm mit eventuellen BASIC-Subroutinen in den Zentralspeicher zu laden und den neuen Programmablauf zu starten.

Als Programmsegmentierungsanweisungen stehen CHAIN- und LINK-Statements zur Verfügung.

⟨Programmsegmentierungsanweisung⟩ 315 295 : = ⟨CHAIN-Statement⟩ 316 | ⟨LINK-Statement⟩ 317 | §§§

Die Zeichenfolge §§§ steht für weitere compilerspezifische Programmsegmentierungsanweisungen.

244 ANSI BASIC, p. 76: "A subprogram may call another subprogram but shall not call itself, either directly or indirectly trough another subprogram."

332.21 CHAIN-Statement

Nach ANSI BASIC gilt für das CHAIN-Statement[245]:

⟨CHAIN-Statement⟩ : =
316 315

CHAIN	⟨Alphanumerischer Ausdruck⟩ 50

Nach ANSI BASIC soll der dem Kennwort CHAIN folgende alphanumerische Ausdruck (programm designator) den Namen eines Files angeben, in dem das aufgerufene BASIC-Programm gespeichert ist;[246] dieses muss nicht unbedingt ein BASIC-Quellenprogramm sein.[247]
Bei Ausführung eines CHAIN-Statements geht das BASIC-Programm, von dem aus das CHAIN-Statement ausgeführt wird, im Zentralspeicher verloren.
Normalerweise wird das CHAIN-Statement benutzt, um grössere BASIC-Programme in kleinere Segmente zu unterteilen, die sich jeweils seriell nacheinander zu einem Ganzen ergänzen.[248]
Bei Kleincomputersystemen reicht daher der begrenzte Speicherplatz durch geschickte Anwendung des CHAIN-Statements selbst für komplexe Anwendungssysteme, wie Finanz- und Betriebsbuchhaltung einschliesslich Fakturierung, aus.

333.22 LINK-Statement

Die Struktur des LINK-Statements ist bis auf das Kennwort mit dem CHAIN-Statement identisch.

245 ANSI BASIC, p. 80. – ANSI BASIC 82, p. 82.
246 ANSI BASIC, p. 80: "In a typical implementation a program-designator will be the name of a file containing that program."
247 ANSI BASIC, p. 80: "The program chained to need not be a BASIC source program."
248 ANSI BASIC, p. 80: "The chain-statement allows separate programs to be executed serially without programmer intervention. Such a facility is useful for segmenting large programs."

⟨LINK-Statement⟩ : =
317 315

LINK	⟨Alphanumerischer Ausdruck⟩ 50

Das LINK-Statement wurde für Nixdorf Business-Basic realisiert. Es unterliegt noch keiner Norm. Im Gegensatz zum CHAIN-Statement führt es dazu, dass alle geöffneten Kanäle, die Werte der Variablen, die Dimensionen der Felder usw. für das aufgerufene Programm erhalten bleiben.[249] Nur der Programmcode des rufenden Programms wird im Zentralspeicher gelöscht, bevor der des aufgerufenen Programms geladen wird. Beim LINK-Statement entfällt dadurch die beim CHAIN-Statement meistens notwendige Wertübergabe zwischen Segmenten mittels Files.
Korrekte Programmsegmentierungsanweisungen stellen die folgenden Statements dar.

```
1000 CHAIN A$
 550 CHAIN "ABRECHNUNG"
2000 LINK "MAHNUNGEN"
```

Mit dem nachstehenden Programm wird ein für den Dialogbetrieb typischer Auswahlkatalog (Auswahlmenü) auf einem Bildschirm erzeugt. Je nach Wahl einer Zahl zwischen eins und sechs verzweigt der Programmablauf mittels eines CHAIN-Statements zu einem der sechs möglichen Programme, die die Namen KUNDEN1, KUNDEN2, KUNDEN3, KUNDEN4, KUNDEN5 und KUNDEN6 tragen. Das vorgestellte Programm ist in dieser Form auf einem System Nixdorf 8870 lauffähig.

```
100 REM AUSWAHLKATALOG
110 REM TEXTVERARBEITUNG NACH NIXDORF 8870/1
115 DIM B$(1), C$(54), D$(7), F$(3), X$(7)
120 LET C$(1,9) ="AUFNEHMEN"
130 LET C$(10,18)="ANZEIGEN."
140 LET C$(19,27)="AENDERN.."
150 LET C$(28,36)="LOESCHEN."
160 LET C$(37,45)="LISTEN..."
170 LET C$(46,54)="BEENDEN.."
180 LET D$="KUNDEN "
190 LET F$="..."
```

249 Nixdorf. System-Software Business-Basic, S. 10.64.

```
200 FOR I=1 TO 6
220   PRINT TAB(5,I);D$;C$(I*9-8,I*9);F$;I
230 NEXT I
240 INPUT TAB(5,I+3),"NUMMER WAEHLEN ",B$
250 LET A = B$
260 IF A < 1 GOTO 240
270 IF A > 6 GOTO 240
280 LET X$ = D$(1,6),B$
290 CHAIN X$
300 END
```

Der angezeigte Bildschirminhalt ist:

```
KUNDEN AUFNEHMEN... 1
KUNDEN ANZEIGEN.... 2
KUNDEN AENDERN..... 3
KUNDEN LOESCHEN.... 4
KUNDEN LISTEN...... 5
KUNDEN BEENDEN..... 6


NUMMER WAEHLEN
```

Fehlermöglichkeiten bestehen beim CHAIN- und LINK-Statement darin, dass der Name des aufgerufenen BASIC-Programms nicht vorhanden ist.

Karl Weber / Carl Wolfram Türschmann

BASIC 1

Lehr- und Handbuch der Programmiersprache BASIC mit wirtschaftswissenschaftlichen Anwendungsbeispielen.

Das in 2. Auflage erschienene Lehr- und Handbuch soll als BASIC-Einführung und zur BASIC-Vertiefung dienen. Der Band orientiert in einem ersten Kapitel über die BASIC-Entwicklung im allgemeinen und vermittelt einen Überblick über eine Reihe von Maschinensystemen, die BASIC als Programmiersprache zulassen. Das zweite Kapitel führt in die Grundlagen der BASIC-Programmierung ein. Im dritten Kapitel wird BASIC unter besonderer Berücksichtigung der BASIC-Elementaranweisungen behandelt. Der metalinguistischen Darstellungsweise von Backus folgend, wird die BASIC-Syntax dargestellt und verbal die BASIC-Semantik erörtert. Zur Illustration der theoretischen Ausführungen dienen zahlreiche Beispiele, die mit dem FOSBIC-Compiler bearbeitet wurden. Das vierte Kapitel vermittelt einige praxisnahe, speziell für Wirtschaftswissenschaftler relevante Beispielprogramme.
Seit Veröffentlichung der ersten Auflage im Jahre 1977 hat BASIC unter allen Programmiersprachen eine dominierende Stellung erreicht. Gefördert wurde diese Entwicklung durch den benutzerfreundlichen Charakter von BASIC im allgemeinen und die fast ausschliessliche Verwendung von BASIC auf Kleincomputersystemen mit Dialogorientierung im besonderen. Die zweite Auflage der vorliegenden Arbeit wurde unter Mitberücksichtigung der neueren BASIC-Sprachentwicklungen weitgehend überarbeitet.

Lehr- und Handbuch der Programmiersprache BASIC mit wirtschaftswissenschaftlichen Anwendungsbeispielen, Band 1. Uni-Taschenbücher 588. 2., überarbeitete Auflage 1982. 274 Seiten, kart. DM 23,80

Karl Weber / Carl Wolfram Türschmann

BASIC 3

Lehr- und Handbuch der Programmiersprache BASIC mit wirtschaftswissenschaftlichen Anwendungsbeispielen.

Dieser neue Band behandelt die Systemanweisungen zur Verwaltung von BASIC-Programmen in benutzereigenen Programmbibliotheken und befasst sich mit den Sonderanweisungen, insbesondere mit Graphik und Diagnosen. BASIC 3 erscheint im Frühjahr 1983.

Lehr- und Handbuch der Programmiersprache BASIC mit wirtschaftswissenschaftlichen Anwendungsbeispielen, Band 3. Uni Taschenbücher 1223

VERLAG PAUL HAUPT BERN UND STUTTGART

Institut für elektronische Datenverarbeitung Zürich (Hrsg.)

FORTRAN

Ein PU-Lehrgang mit Programmbeispielen für Ingenieure, Techniker, Ökonomen, Naturwissenschaftler

„IDV Lehrprogramm"
Band 1: Basistext und Lösungen
Band 2: Steuertext und Übungen

Zusammen 212 Seiten, Spiralheftung Fr. 68.–

Karl Weber / Carl Wolfram Türschmann

FOSBIC-Compiler

Ein BASIC-Compilersystem auf FORTRAN-Basis.

„BASIC-Software" 1. 1977.
224 Seiten mit vielen Zeichnungen und Compiler-Ausdrücken, kart. Fr. 40.– / DM 31,–

Karl Weber

BASIC für Anfänger

„BASIC-Software" 2. 1979.
153 Seiten mit 17 Abb. und 4 Tab., kart. Fr. 28.– / DM 31,–

Karl Weber

BASIC Programmier-Richtlinien für Austauschprogramme

„BASIC-Software" 3. 1979
81 Seiten, kart. Fr. 30.– / DM 33,–

VERLAG PAUL HAUPT BERN UND STUTTGART

„Institut für Informatik der Universität Zürich" (Hrsg.)

EDV-Pflichtenhefte

Wegleitung für die Erstellung von EDV-Pflichtenheften

Computer-Beschaffungsprojekte sind anspruchsvoll und aufwendig und verlangen beträchtliche Investition an Geld und Arbeit. Im gesamten Beschaffungsorgan eines Computers oder einer Lösung ist das Pflichtenheft deshalb ein bedeutendes Element. Es muss ein Maximum an relevanten Informationen bieten. Aufbau und Substanz des Pflichtenheftes bestimmen sehr stark die Aussagekraft und den Wert der Offerte. Auch wird das Gelingen des geplanten Projektes nicht unwesentlich beeinflußt.

„Schriftenreihe des Instituts für Informatik der Universität Zürich" 4. 1981.
173 Seiten, kart. Fr. 25.80 / DM 28,–

„Institut für Informatik der Universität Zürich" (Hrsg.)

EDV-Kennzahlen

Praxisbezogenes Instrumentarium zur Beurteilung der EDV-Wirtschaftlichkeit.
Mit Beiträgen von Walter J. Bruhin, Felix Brunner, Richard Heinzer, Hans Peter Koch, Gabriel G. Minder, Peter Nagel, Peter Nussbaumer, Reinhard Schretter jun., Edgar Vöhler, Heinz Waldburger.

„Lohnt sich unsere EDV eigentlich?" Um die für den Einsatz der EDV Verantwortlichen bei der Beantwortung dieser Frage zu unterstützen, hat die SVD ein Kennzahlen-System erarbeitet, das für eine betriebseigene differenzierte und dachgerechte Beurteilung der Wirtschaftlichkeit, getrennt nach Kosten, Leistung, Nutzen, der EDV bildet.

„Schriftenreihe des Instituts für Informatik der Universität Zürich" 2. 2. Auflage 1981.
122 Seiten, kart. Fr. 24.50 / DM 27,–

VERLAG PAUL HAUPT BERN UND STUTTGART

Bruno Graber

Computergestützte Informationssysteme in kleineren bis mittleren Unternehmungen

„Schriftenreihe des Instituts für Informatik der Universität Zürich" 1. 1978.
219 Seiten, kart. Fr. 32.– / DM 35,50

Thomas Fischer

Computer-Kriminalität

Gefahren und Abwehrmassnahmen

„Betriebswirtschaftliche Mitteilungen" 71. 1979.
43 Seiten mit 10 graphischen Darstellungen, kart. Fr. 10.80 / DM 12,–

Peter Basci

Operationale Informationssystem-Gestaltung

(OISG) Vorschlag für eine computergestützte Informationssystem-Gestaltung

„Betriebswirtschaft der Hochschule St. Gallen" 5. 1978.
383 Seiten, kart. Fr. 69.– / DM 74,–

Hansbert Suter

Die langfristige Planung von computerunterstützten Informationssystemen

Dargestellt am Beispiel einer Handelsbank

„Bankwirtschaftliche Forschungen" 62. 1980.
162 Seiten mit 33 graphischen Darstellungen, kart. Fr. 28.– / DM 31,–

VERLAG PAUL HAUPT BERN UND STUTTGART

Das vorliegende Lehr- und Handbuch dient der BASIC-Einführung und der BASIC-Vertiefung. Während in Band 1 die BASIC-Elementaranweisungen behandelt wurden, befaßt sich Band 2 mit den BASIC-Sonderanweisungen. Matrizen-, File- und erweiterte Programmsteuerungsanweisungen werden unter Bezugnahme auf die Standardisierungsvorschläge von ANSI BASIC dargestellt. Zur Illustration der theoretischen Ausführungen dienen zahlreiche Beispiele. Das Buch vermittelt weiterhin einen Überblick über Minicomputersysteme mit BASIC-Prozessoren.

UTB Wirtschaftswissenschaften
Informatik / Datenverarbeitung

ISBN 3-258-03175-4